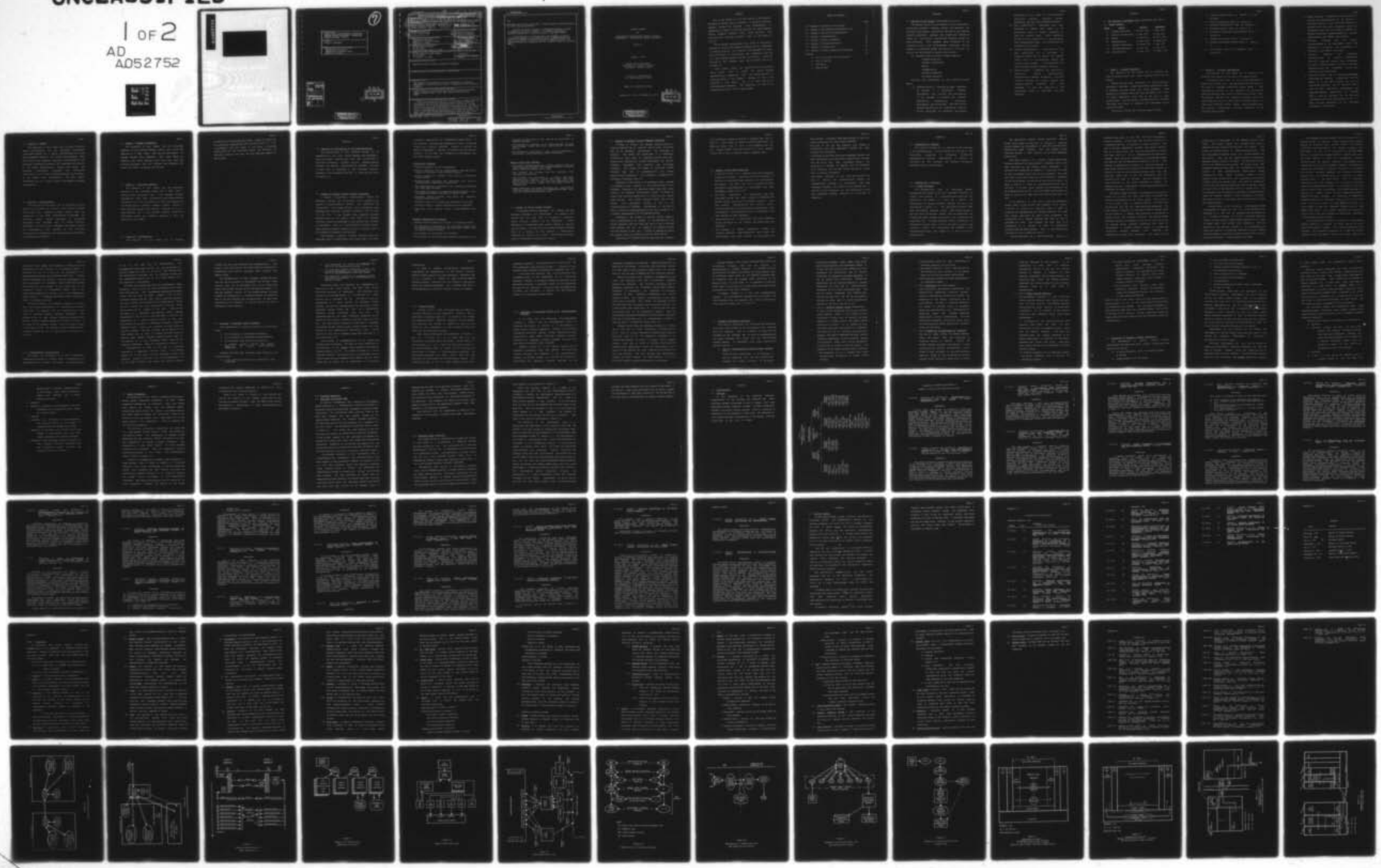
AD-A052 752 KANSAS STATE UNIV MANHATTAN DEPT OF COMPUTER SCIENCE F/G 17/2
PROJECT REPORT FOR FUNCTIONALLY DISTRIBUTED COMPUTER SYSTEMS DE--ETC(U)
FEB 78 V E WALLENTINE DAAG29-76-G-0108

UNCLASSIFIED ARO-13835.1-A-EL-PT-2 NL

1 OF 2
AD
A052752



7

4. TITLE *(and Subtitle)*

PROJECT REPORT FOR FUNCTIONALLY DISTRIBUTED COMPUTER SYSTEMS DEVELOPMENT: SOFTWARE AND SYSTEMS STRUCTURE. PART II

7. AUTHOR(s)

Virgil E. Wallentine

9. PERFORMING ORGANIZATION NAME AND ADDRESS

Kansas State University
Department of Computer Science
Manhattan, KS 66506

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered) (9) Final rept. 1 Nov 76-25 Jan 77

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER (18) ARO (19) 13835.1-A-EL-PT-2	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) (6) PROJECT REPORT FOR FUNCTIONALLY DISTRIBUTED COMPUTER SYSTEMS DEVELOPMENT: SOFTWARE AND SYSTEMS STRUCTURE. PART II,		5. TYPE OF REPORT & PERIOD COVERED Final January 15, 1976 NOV ~~October 31~~, 1977
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) (10) Virgil E./Wallentine		8. CONTRACT OR GRANT NUMBER(s) (15) ✓ DAAG 29-76-G-0108
9. PERFORMING ORGANIZATION NAME AND ADDRESS Kansas State University Department of Computer Science Manhattan, KS 66506		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS (12) 100p.
11. CONTROLLING OFFICE NAME AND ADDRESS US Army Research Office P O Box 12211 Research Triangle Park, NC 27700		12. REPORT DATE (11) 7 Feb [redacted] 78
		13. NUMBER OF PAGES 93
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) US Army Computer Systems Command Attn: CSCS-AT Ft. Belvoir, VA 22060		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

The findings in this report are not to be construed as an official Department of the Army position, unless so designated by other authorized documents.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Computer Networks, Mini/Microcomputer Networks, Distributed Processing Computer Message Systems, PASCAL, DBMS, Back-end System, MIMICS, KSUBUS

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This is the second and final part of the report of research performed by Kansas State University in multiple processor computer systems and and networks. Part I covered the Design Phase of the effort; and this report covers the follow-on implementation, integration, test, and demonstration of a prototype model of the network. The network model consists of a cluster of minicomputers and microcomputers with supporting software. The model has been named MIMICS (MIni-MIcroComputer System) and uses a high-speed vendor-independent data bus, named KSUBUS, that was designed,

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

391 123 [illegible]

(20)

developed, and built for this network. Network hardware included Interdata's 85, 7/32, 8/32 and IBM 370/158.

The principal network software is a message system which is capable of residing in a variety of computers. The hardware independence is achieved by design and by coding the software in Concurrent PASCAL.

A specification of a distributed data base management system was developed and implemented in the MIMICS network. A DBMS named TOTAL was used in the prototype. The general problems of DDBMS were studied, and solution syntheses are presented as well as a simulation model for a back-end DBMS.

PROJECT REPORT

for

Functionally Distributed Computer Systems
Development: Software and Systems Structure

PART II

February 7, 1978

KANSAS STATE UNIVERSITY
Department of Computer Science
Manhattan, Kansas 66506

Project No. P-13835-A-EL
U.S. Army Research Office

GRANT NO. DAAD-29-76-G-0108

January 15, 1976 to October 31, 1977

PREFACE

This is the second of a two part report of the research performed by Kansas State University in multiple processor computer systems and networks. This investigation is supported by a grant of $190,000 from the U.S. Army Research Office, Research Triangle Park, North Carolina. The University has provided matching funds in the amount of $28,383.

The principal investigator is Dr. Virgil E. Wallentine, assisted by faculty and graduate students of the Department of Computer Science. The research was performed at Kansas State University in coordination and cooperation with the U.S. Army Computer Systems Command, Fort Belvoir, Virginia. The term of the research grant was 15 January 1976 to 31 October 1977.

Part I report covers the research effort through the Design Phase. Part II covers the effort through implementation, integration, test, and demonstration of a prototype model of the network. Chapters 1.0 and 2.0 are identical in both Parts I and II so that they can be self-contained documents. The appendices of Part II are extensions of Part I for the same reason.

TABLE OF CONTENTS

CHAPTER 1

1. Overview of the Project (replicated from Part I)

The general nature of the research is the investigation of multiple processor computer systems and networks. The Principal Investigator, assisted by the faculty and graduate research assistants, explored the alternative methods of design of a functionally distributed computer network for data processing. This research takes advantage of the potential of mini- and microcomputer technology. The end product is a prototype system that serves as a test bed for testing the performance of typical data systems.

The research effort followed a phased approach:

Problem Definition
Solution Alternatives
Design
Implementation
Systems Integration
Prototype Operation

The work was concentrated in four specific problem areas:

(1) Software Utility - Software has been developed to operate in a multivendor computer environment. This involved the investigation of the problems of multiple CPU software portability, adaptability, conversion, development, and maintenance. This area of the program concentrated on a comparative analysis of the techniques for achieving the desired

portability in the areas of data processing application programs, operating systems dependence, and data base management systems.

(2) Data Accessibility - Techniques have been developed to permit data bases to be distributed across a network accessible to local and regional query. These techniques provide for the protection of the data bases from unauthorized access. This distribution is transparent to the user.

(3) Hardware Specification - Specifications have been established for a high-speed (10 megabytes/second) bus to permit off loading (onto a mini- or microcomputer system) the network control system. A prototype has also been developed between Interdata machines.

(4) Network Control - Research has been done on alternative network configurations. Communications, message processing, and other controls required for system balance and hardware/software interface have been developed. To test the viability of the distributed system, a prototype has been developed.

CHAPTER 2

2. The Technical Development Plan (replicated from Part I)

2.1 Phase Schedule

Phase	Effort	Started	Completed
I	Problem Definition	15 Jan. 1976	1 Feb. 1976
II	Solution Alternatives	2 Feb. 1976	14 May 1976
III	Design	15 May 1976	15 Nov. 1976
IV	Implementation	16 Nov. 1976	14 March 1977
V	Systems Integration	15 March 1977	1 July 1977
VI	Prototype Operation	2 July 1977	1 Aug. 1977
VII	Documentation	15 Aug. 1977	14 Sept. 1977

2.2 Phase I - Problem Definition

The objective of this phase was to identify the specific problem areas upon which research effort must be applied.

Certain problems were identified in the KSU proposal submitted to the Army. There were several alternative areas of research outlined along with those problems. These problems and research areas were then reviewed in the light of recent experience, both here at KSU and in other universities. From the review, new definitions of problems were made and areas of research were described in more definitive terms.

The review covered the following specific areas:

a. Software Utility (Para. II, Section A of the Proposal)

b. Data Accessibility (Para. II, Section B)

c. Hardware Specification (Para. II, Section C)

d. Resources Control (Para. II, Section D)

e. Mini/Micro Technology Considerations (para. V, Section B)

f. Hierarchical Systems (Para. V, Section B)

g. Dynamic Performance Monitor (Para. V, Section C)

h. Distributed Control of Networks (Para. V, Section D)1

2.3 Phase II - Solution Alternatives

The objective of this phase was to establish the direction and approach to be used in the design effort.

Consideration was given to the various approaches to achieve the project goal. The approaches were considered in the light of problems identified during Phase I. This consideration resulted in the selection of the research direction believed most likely to achieve the project goal. The selected approach and research direction were to be coordinated with the U.S. Army Computer Systems Command, to insure that follow-on design effort would be appropriate. The solution alternatives provided direction in the following design areas:

a. Network Topology: Alternative configurations of hardware and communications to be tested in a prototype solution. Mini- and microcomputers in networks with large scale computers were to be considered along with the attendant problems of software interface, message processing, network communications and network load balance.

b. Software: Software with and without microprocessors to be considered, along with portability, conversion, and maintenance of software in the network environment.

c. Hardware: Technical and functional characteristics of various commercially available computing systems were considered. The use of microprocessors to achieve hardware compatibilities was considered, along with reliability and performance.

d. Data Base System: Use of data base management techniques was proposed. Alternatives included design of a new DBMS suitable for use in distributed networks, use of existing DBMS´s such as IDMS (Cullinane Corporation), and hardware/software modification of existing DBMS´s. Solution synthesis was to take into consideration the long term desires of the U.S. Army and the constraints of the resources available to the project.

2.4 Phase III - Design

The objective of this phase was to write a detailed design specification for the prototype functionally distributed network and a design specification for a data base management system to run within the prototype. Using the direction and approach developed during Phase II, the Principal Investigator directed the developing and writing of the design specification. During this phase, the Principal Investigator published the "Functional Specification" (CS 77-04) to provide guidance and continuity to the several Chief Investigators. The DBMS design development and the network design development proceeded independently.

2.5 Phase IV - Implementation

The objective of this phase was to construct and test modules and document the various system designs produced during Phase III. Graduate students working under the supervision of the Chief Investigators programmed the prototype system. Modern techniques of structured programming were followed and all effort carefully documented. As modules were completed and tested, results and documentation were reviewed by the Principal Investigator for compliance with basic design and conformity to documentation standards.

2.6 Phase V - Systems Integration

The objective of this phase was the integrated operation of the prototype message system test bed and the operation of a DBMS within that test bed environment. As systems modules were completed, they were tested in operation with other modules using both test data and synthetic programs. When all modules were completed, the system was tested, corrected, and refined.

2.7 Phase VI - Prototype Operation

The objective of this phase was the successful operation of the prototype system and a DBMS under a synthetic data processing load. File transfer protocol programs and a DBMS were distributed across three machines (Interdata 7/32, 8/32, and IBM/370) to test the viability of the communications software (message system). Finally, a small operating system (SOLO [PBH 76]) (modified to run in a small machine) and a line printer spooling system were distributed across two Interdata machines to test the operation of the KSUBUS.

2.8 Phase VII - Documentation

The objective of this phase was to complete documentation of the prototype system. Copies of reports by the individual investigators were provided to the U.S. Army in accordance with the terms of the grant. A complete specification of the prototype system was prepared for delivery as part of the set of final technical reports of the project.

CHAPTER 3

3.0 Overview of Contributions to the State-of-the-Art

The contributions of this research project can be categorized in two basic areas--research and development. They are further characterized into work in the areas of data base management, software utility, computer network software, and computer interconnection hardware. The work in each area is described in the following sections. Abstracts of the documents produced in each area are given in Section 7.0.

3.1 Summary of Network Control Software Activities

A network interprocess communication system was designed, developed and constructed with a capability to support a resource-sharing network. Its properties are listed below. This system contains a set of protocols which encompass the currently available protocols [WAL 72] [CEK 74] [FAL 72]. A new concept which supports multiple physical lines on one logical line [RAW 77] in a message switching system was developed in this project and included in the system. This is a fault-tolerant software concept which permits automatic recovery of data on logical lines in the presence of physical line failures.

New concepts in function oriented protocols were also developed [WJH 77] which permits the novice user to be naive in his/her application of interprocess communication. It also permits a sophisticated programmer to treat slow-speed lines as the critical resource. Finally, it provides the system's programmer a resource allocation and distribution protocol. All of these new concepts are incorporated into the MIMICS message system.

Architecture Features

- Network computers can be minicomputers
- Network computers can be heterogeneous machines (which support a Concurrent PASCAL virtual machine)
- Control software is portable (via porting of Concurrent PASCAL system)
- Message system functions are off-loaded into mini- or microprocessor communication controllers
- The communications controllers are physically protected from user level software
- High-speed data paths (10 megabytes/second transfer rate) are supported between machines in a close cluster
- Low-speed, packet switched data paths are supported between remote machines
- Different types of low-speed line protocols can be used
- Different types of network topologies can be supported
- Data paths allow back-up routes in case hardware lines fail

Software Implementation Features

- Network control software is written in Concurrent PASCAL
- The software is guaranteed to be free from a large class of type-conflict and run-time synchronization errors that can plague systems software
- The software is structured to be portable
- The software is well structured, modular, and easy to read

- Its size is very close to the size as its counterpart in assembly language

- The software is intended to be fault tolerant (of both user software faults and data communications hardware faults)

- The software is structured so that it can be expanded or contracted to suit particular network structures

__Message System User Features__

- The network message system has a simple interface with the local operating system in each network computer (in case a network operating system is not used)

- User programs are isolated from all low-level data transmission protocols

- User programs are identified by a logical name which specifies their machine, unique task name, and a user defined optional name; and two tasks can communicate when they know each others´ names and both agree on mode of communication

- Basic operations are connect/disconnect and send/receive; different options are allowed to accommodate either naive users or system-programmer-type users

3.2 __Summary of MIMICS Network Hardware__

A high-speed vendor-independent bus (KSUBUS) has been designed, developed, and constructed. It supports a new concept [MAC 78A] [MAC 78B] of "clusters" of computers. Within clusters, KSUBUS´s can be "pipelined," without degradation of performance when larger clusters of machines are configured. The bus moves data memory-to-memory between hosts in a cluster and thus permits a communication controller to contain the entire (off-loaded and secure) network interprocess communication system.

3.3 Summary of Software Utility Research Activities

In order to utilize the message system software on multiple machines and in multiple environments, the system must be as portable, maintainable, and adaptable as possible. The system was constructed in a hierarchy, isolating at each level an adaptable function [WJH 78B]. Several such adaptations include high-speed bus control (KSUBUS), slow-speed line protocol, and off-loaded functions. The system was coded in Concurrent PASCAL (CPASCAL) [PBH 75A] to accommodate--to the degree possible--both properties. The language´s properties are reviewed in reference [VEW 76].

Adaptability was tested in an experiment which assumed no knowledge of the language or the system written in the language. Students in an operating system class were able, within a one semester three-credit course, to adapt the single user operating system SOLO to a batch and two remote input systems. This effort required about four (4) man months. This phenomenal success is due to the understandability of CPASCAL. Maintenance in our view is also a function of understandability and thus a property of a CPASCAL implementation of the message system.

Portability was tested by porting Brinch Hansen´s PDP-11 implementation of CPASCAL to the Interdata 16- and 32-bit computers [DNN 76A] [DNN 76B] [DNN 77] and to the NCR 8250 computer [DM 78]. In summary, all systems software must be adapted in some way. CPASCAL is adapted by coding the kernel, which is a three to four man month effort.

Productivity of system code and code size was measured in an experiment comparing CPASCAL to assembly code [JRR 78] [REW 77]. Code written in CPASCAL produced effectively the same size object code as that in the assembler code [RAY 77]. Correct CPASCAL code was produced in 30 to 40 percent of the time it took to produce correct assembler code.

3.4 Summary of Data Base Activities

1. A simulation model of a back-end data base management system was developed. This model was later enhanced to describe a distributed DBMS operating in a multicomputer environment. The model was used to analyze critical performance characteristics of distributed data base management systems [CS 76-12].
2. The basic organization of distributed data base management systems has been studied to determine the proper software structure for processor modes in a data base network. Configurations with multiple hosts, multiple back-ends and bi-functional machines have been investigated. The information flow in distributed data base systems has been specified.
3. Specifications for a distributed data base management system conforming to the CODASYL philosophy have been developed.
4. The problems of memory management, rollback and recovery, deadlock, user-transparent data access and data movement have been studied for distributed data

base systems. Procedures have been defined to cope with these distributed data base design problems.

5. The state of the art and industry with respect to distributed data base management systems has been surveyed and chronicled.

6. A prototype distributed data base management system has been implemented using the MIMICS communication software and the TOTAL data base management system. The prototype is intended to operate with the Interdata 8/32, Interdata 7/32, and IBM 370/158 serving as either host or back-end processors.

7. The overall objective of this data base research has been to explore the organizational, design, and implementation problems of distributed data base management systems. The intent of this research was to provide a solid foundation for the realization of data base management systems operating on a network of computers.

CHAPTER 4

4.0 Implementation Approach

This section consists of an overview of the structure of the implementation. It also contains an overview of the portability properties of Concurrent PASCAL--the implementation language. Additionally, it contains an evaluation of the software and hardware engineering principles used in this project.

4.1 Distribution of Software

4.1.1 System Structure

The implementation must, as previously stated, accommodate the message system (MS) (exchange) functions in either a host computer or a communications controller (CC) attached to the memory of a host via a KSUBUS. The distribution of software processes between host and CC are illustrated in Figure 4.1. The application program, DBMS task, and Network Resource Control (NRC) always reside in the host while the message system (and its subsystems, the cluster and packet system) may reside in the communications controller (CC). In these diagrams, control information is indicated by single lines (<----->) and data by double lines (<=====>). Figure 4.1a illustrates the elements of the implementation which must reside in a host if it has no associated CC.

The application program issues high-level message system calls (these are specified in Section 3). These MS calling formats are interpreted by the user envelope which synchronizes these requests with the message system processes through a SYSQ function in the local operating system (LOS).

This SYSQ function is a System Control Block (SCB) exchange mechanism for sophisticated processes such as the user envelope and the message system processes. Its functions are documented in Appendix 2, along with its usage by both the message system and the user envelope. The SCBs are fixed in size and contain all the information necessary for the MS processes to execute the user level MS calls. As shown in Figure 4.1, SYSQ is incorporated into the LOS of the host. It is SYSQ which permits the off-loading of the MS processing onto the CC. This off-loaded configuration is shown in Figure 4.1b, where one CC is serving more than one host.

It is important to note that a data base management system (DBMS) is just another task (process) in our system, a result of viewing all functions (such as DBMS) as resources and implementing full resource sharing. The DBMS task issues high-level MS calls whose parameters are interpreted by a system envelope and sent to the MS processes via SYSQ. This permits the DBMS task to be implemented in a variety of ways to accommodate multiple-vendored DBMS systems. That is, the standardized interface is the MS functions (calls).

The NRC functions as a task controller. That is, it controls those tasks in the host which are considered as network resources. It does both file and task allocation to a requested service. For example, it might allocate a DBMS task to service application program data access requests. That is, it could start the task and manufacture a network task name (C.M.T.P.) to be referenced by the application program. From that point, messages can be exchanged between the DBMS task and the application program via the MS. This protocol is exemplified in Figure 4.2. NRC also contains a supervisory function for unusual conditions--for example, an overflow of SCBs in SYSQ directed towards a task. The SYSQ redirects these SCBs to NRC for processing under the assumption the task is aborted. NRC will implement only these functions in the first prototype of MIMICS. Its expanded role in a *distributed* network operating system is documented in report CS 77-4.

The basic system structure is hierarchical, as shown in Figure 4.3A, in that the user level task is isolated from network considerations by the lower levels. Each level is in turn a lower level of network protocol. The user task and envelope and the NRC were previously described. The SYSQ monitor (a monitor is a Concurrent PASCAL concept which can be viewed as a shared data structure for processes to access) is the SCB exchange facility. The Message System Processes receive MS requests in the form of SCBs from SYSQ. The message system processes coordinate with their MS counterparts in another network machine to synchronize connection, command transfer, data transfer, and disconnection between user processes. The status of this coordination is maintained in the Message Monitor (MESS TABLE). The Cluster System (CS) processes will exchange commands and data between host memories in a cluster. They access the message monitor to update message status. They move data memory-to-memory across one or more KSUBUS's and data movers. They use the packet monitor to synchronize commands between nodes in a cluster. The data mover drivers control the KSUBUS and data movers. The Packet System (PS) processes coordinate with their PS counterparts in another machine to move packets--error free--across remote transmission lines. These processes support multiple physical lines (controlled by the synchronous line control drivers) within one logical line. This permits reliability of the link, and recovery procedures are not neccary except for loss of the final physical line on a logical line. The PS processes also coordinate the exchange of command packets (send and receive requests) among machines in a cluster. The Packet Monitor is the shared data structure which stores and routes packets.

Further detail on the processes in the message system is presented in Figures 4.4 and 4.5. The MS processes are presented in 4.4 and the PS processes are illustrated in 4.5. As shown in Figures 4.4 and 4.5, processes typically access a monitor and are blocked there waiting for an event to happen which will resume their execution. The data accessed in the monitor (shared by all processes which access the monitor) is typically operated on by the process and then placed in the same or another monitor (as an event) for another process. It then repeats the cycle.

The MSSENDREC process accesses SYSQ for requests on the message system. On arrival, these requests are validated against the connection table entries MSCONN. If a request is active on this connection, the request is queued in the MESS TABLE. If not, it is made active in the MESS TABLE which is the event that resumes MSSTART. MSSTART then calls the PACKET MONITOR to start the "handshaking" protocol (matching send and receive requests) with MSCMDRCV which coordinates the matching of MSSENDs, MSRECVs, and MSAUCs and updates the MESS TABLE to reflect a "matching." The INMOVER and OUTMOVER processes assemble and disassemble messages, respectively. The OUTMOVER awaits on a "matched" or "ready" message status in the MESS TABLE. Then it repeatedly disassembles the user message into packets and transmits them via the PACKET BUFFER (PBM). It only waits in the PBM when no buffer space is available. The INMOVER process also awaits on a "ready" status and then repeatedly assembles packets into user messages. On completion of INMOVER and OUTMOVER functions, a "complete" status is set in the MESS TABLE. This is the event on which MSDONE is waiting. It then resumes to clear this "complete" request from the MESS TABLE, make "active" any queued requests, and use SYSQ to send an SCB to the user envelope to indicate the completion of this message system request.

The PS processes are shown in Figure 4.5. Each process either transmits or receives (or both) data to or from a remote line control driver. This driver governs the line protocol. The procedure on sending is to get full buffers (packets) from the packet monitor and pass them to the driver and then repeat the procedure. These drivers are implemented in some "kernel" of an operating system. It could be the LOS of a host or the multiplexor of an implementation language (see reference [DNN 76A]). Several types of processes are possible. The only requirements are that the protocol be consistent with the PBM entry points (functions) and the protocol on the other end of the connection. Figure 4.5 illustrates logical line 1 as a user-defined protocol and logical line 2 as a prespecified window protocol. It supports full-duplex (FDX), half-duplex (HDX), and simplex protocols.

The window manages the sequence numbers on a logical line. This enables maintenance of synchronism of sequence counts when one physical line on a logical line is lost (with no explicit recovery procedure necessary). Each process can be of type listener (receiving packets from a driver), xmitter (sending packets to a driver), or both. In FDX there is one listener process and one xmitter process. In HDX there is only one process of type listener/xmitter. A simplex process can be of type xmitter or listener, but not both.

4.1.2 Implementation Considerations

The system was structured as a set of (concurrent) cooperating processes to enhance ease of construction and understanding and to permit off-loading of these processes to run in the host and CC simultaneously. The implementation thus places the MS, PS, and CS processes in the host (under the LOS) or in the CC. This implies that the implementation language must be portable to hosts and CCs, implemented under an LOS or on a "bare" machine (CC), and supportive of concurrency.

At least two alternatives exist for implementing these processes. First, the MS, PS, and CS processes can be tasks under the host LOS. The interprocess communication system of the LOS can then be used. However, this is not a standard interface and the task switch times are typically too long. Second, the processes can be pseudo-tasks within the CC or within one task (partition) of the LOS. These pseudo-tasks can be created in two ways. These can be processes in a high-level multitasking (HLMT) language implemented in the CC or as a host LOS task; or they can be implemented by a simple tasking monitor in the same environment but written in a sequential language (typically assembler level). Both can be achieved by first coding in the HLMT language. Then the language can be ported or the coded version can be used to develop low-level coding in what has been termed "reliable machine coding" [PBH 77]. Our approach is to use Concurrent PASCAL which has the necessary properties. These are illustrated in Section 4.2.

Implementation of SYSQ in the host is typically achieved as a supervisor function (SVC). If the message system is in the host, the MS processes access SYSQ directly. However, if MS, PS, and CS are implemented in a CC, the SYSQ function must cross the host-CC boundary.

Figures 4.6A and 4.6B illustrate this implementation. SYSQ exists in both host and CC, and these SYSQs are connected by asynchronous line control processes which transmit SCBs between SYSQs.

The implementation of data movement between the user task and the MS processes is implemented via the data mover control. When the MS processes are in a CC, these data movers are hardware devices which move memory-to-memory. In the host resident version of the message system, the data movement is performed by a software movement (an executive task in the LOS) between host LOS partitions. No other code modifications are necessary to utilize the MS, PS, and CS processes in both versions.

4.2 Language - Concurrent PASCAL (CPASCAL)

The implementation language was chosen on the following bases:

1. It is conducive to structured programming
2. It is conducive to structured multiprogramming
3. Its structure lends itself to portability
4. It supports dynamic linking and overlay capability (to provide reconfiguration capability)

CPASCAL (see [PBH 75B]) satisfies these criteria in the following ways:

1. It has high-level control and (extensible) data structures

2. It is concurrent by nature with monitors (see [CAH 74]) to synchronize processes

3. Its concurrent process multiplexor is small and it can be implemented on a bare machine or as a task in a multiprogramming operating system

4. The loading and execution of sequential PASCAL (see [J&W 74]) programs can be controlled in a CPASCAL process.

Concurrent PASCAL was designed and implemented on a PDP-11/45 by Per Brinch Hansen (see [PBH 75B]) at the California Institute of Technology. KSU personnel have "ported" this language system to an Interdata 8/32. Other portings are in progress [MB 76]. The PDP-11 version is implemented on a "bare" machine (no operating system present), and the 8/32 implementation has Concurrent PASCAL processes running within one task of a multitasking operating system (OS-30/MT) (see [Int C]). The implementations by the Naval Underseas Laboratory are on "bare" 16- and 32-bit Interdata machines (see [Int A and B]). The "bare" machine version has been used to implement the message system, packet system, cluster system, and SYSQ monitor in the communication controllers (CCs). The OS version will be used to implement these same functions in a host which has no CC.

This variety of implementations is the reason that CPASCAL must be portable. The basis for this portability is the smallness of the kernel which supports multiplexing, synchronization, and I/O. It is about 8K bytes on a 16-bit Interdata machine. The portability of the language machine (PASCAL stack machine) depends on the kernel, and the small kernel can be ported by an interpreter or by compiler modification.

In order to support documentation understanding, maintenance, and upgradability on the network software, several CPASCAL manuals have been generated. Reference [VEW 76] is a tutorial of CPASCAL as used in simple network software modules. Reference [WJH 76] contains a tutorial on PASCAL for FORTRAN programmers, and reference [DNN 76B] is the documentation of the 8/32 "ported" version of CPASCAL.

4.2.1 Porting CPASCAL

In order to port CPASCAL, the kernel must be coded in a low-level language on the destination machine. This is described in reference [DNN 76A]. In addition, the CPASCAL and Sequential PASCAL (SPASCAL) compilers must be ported. Both compilers are written in SPASCAL so that only the kernel and the interpreter or code generator need to be coded for the destination machine.

These porting strategies are illustrated in Figures 4.7 and 4.8. In Figure 4.7, it is clear that only the interpreter and kernel need to be coded. Porting PASCAL to the Interdata 8/32 was a four man month effort [DNN 77]. The second strategy shown in Figure 4.8 is to utilize the first seven (7) passes of the modularized compilers. Two additional strategies are clear. First, only the code generator can be targeted for the destination machine. This strategy was used at KSU to "port" CPASCAL to the 16-bit

Interdata computers. The alternative is to code macros for the intermediate code produced by pass 7. A version of this portable macro language is presented in reference [HMF 76].

It is clear that compiled code is more efficient than interpreted code if the interpreter is at a user level language. However, if microcode storage is available on the destination machine, a microcoded version of the interpreter may well be more efficient. Since all these implementations of PASCAL are available, it permits the most efficient choice for a particular target machine.

4.2.2 Evaluation of Concurrent PASCAL as an Implementation Language

It is clear that the structured multiprogramming concepts in CPASCAL rule out time-dependent errors by extending the concept of monitors to a true hierarchy of access rights to system components. Further, its structured programming constructs enhance correct sequential program construction. Using these high-level language concepts, production of "debugged" systems programs is higher than when using an assembler level language.

CPASCAL´s utility as a job control language can be extended to languages other than SPASCAL. The particular changes necessary are dependent on the implementation. However, the minimum change is to modify the interface to accommodate the operating services required by the sequential programs to be executed. These functions must be supplied in the kernel. In the case that the kernel resides as a task under a local operating system (LOS), the service calls to the LOS must be mapped into the interface routines.

There will always exist assembler programmers who can write more efficient code than a high-level language compiler can generate. The low-level constructs such as addresses, registers, and interrupts permit great freedom. Addresses, in particular, are important to efficient accessing of data. Addresses (references) are copied instead of data. The absence of references is the single worst fault of CPASCAL. This forces two processes to copy any data they need to exchange from the private data of the source process into shared data in a monitor and then into the private data of the destination process. This is considerable performance degradation from passing pointers. Such use of references could be incorporated in CPASCAL if they are used in a controlled manner [SIB 76].

Awkward coding forms are sometimes necessary in CPASCAL due to the lack of "trap" facilities (on conditions in PL/I). That is, in many instances an asynchronous condition may occur (such as an I/O completion interrupt or an unsolicited message) to which a process should respond quickly. This case is coded in CPASCAL as an auxiliary process which specifically waits for the condition (in a monitor or the kernel) and then transmits the condition to the user process. The user process must periodically inspect the common monitor (an event monitor). It is not yet clear whether language modification is warranted.

Several changes to the current implementation should be incorporated (however, they are not critical to implementation of network systems). A software time-out facility must be added to the kernel for those devices (such as synchronous lines) which do not present interrupts when an excessive time has elapsed since initiation of an external event. We also think it is important to take the fixed level of priorities out of the kernel and permit queue priorities to be set from a process.

In summary, CPASCAL is a very good implementation language. The modifications suggested are not critical, and more experience with the language will help to isolate any other possible changes [SIL 77] [LOH 77].

4.3 Software Engineering Evaluation

This section summarizes the decisions and techniques which had a major effect on the structuring of the software which was produced. This is intended to point out the strengths of the software and to serve as recommendations for future software work within the U.S. Army Computer Systems Command. In retrospect five items stand out as positive factors:

1. Use of a well-structured program design language (PDL)--

   Since the word "structured" is so overused, it must be explained that the PDL included many features not always included in so-called structured languages. Among these were closed module specifications with parameters tagged as to type and entry/return mode, PASCAL-like data structures and definitions, use of monitor structures to synchronize multiple processes in use of shared data structures, PASCAL-like pseudo-code but with limited nesting of control structures and with use of "multiple-exit" statements so as to limit duplicated code. The PDL was easily translated into CPASCAL code. Weak points were that the PDL was not formally defined and there were no tools available for processing the PDL in any way at all.

2. Use of a <u>top-down</u> <u>modular</u> <u>design</u> methodology-- In spite of the widespread attention to top-down design methods, there is the pitfall in real situations of being able to identify low-level modules of a system before the overall system structure is evolved. As an example of this, note that often when some network is presented, the focus will be on low-level line protocols. However, that type of approach may make network functions subservient to line protocols instead of vice-versa. After some initial sputtering, we were able to develop a top-down design. The benefits of the design approach were as follows:
   --The structure is in a layered form which promotes easy off-loading of levels of the software

--The structure allows for easy interchange of low-level physical line protocols

--The modular form will allow restructuring of the software to form variants of the network which are fit for particular applications

--The structure allowed prototype implementation in incremental stages

3. Use of walk-thrus to review design--
Like structuring, the concept of walk-thrus has received much superficial attention. However, without a well-defined structure for review and acceptance of the design, walk-thrus can be quite useless. We used a structure related to the method of informal proofs of programs, namely the identification and justification of assertions about the program execution established at various break-points. This was moderately successful, although one major error in synchronization across machines was detected after the walk-thrus.

4. Use of CPASCAL as an implementation language--
This was an ideal choice for several reasons. First, the design code was structured to translate easily to CPASCAL. Second, the CPASCAL compiler provides verification of type constraints and process synchonization far in excess of that provided in any other existing compiler. Hence, we can guarentee the network software to be free from a large class of errors

which are detected by the compiler. It is interesting to note that it required considerable effort to get all the network software in CPASCAL to compile; but once compiled, integration and run-time checkout was relatively easy. Third, the CPASCAL listings, augmented with various access and data structure diagrams, serve as a good level of documentation.

5. Use of highly skilled personel--
The programmers consisted of just a few very skilled post-MS students, each working only part time. All were familiar with CPASCAL, process structuring and issues relating to correctness of programs. It is doubtful if this project could have been completed using "entry level programmers."

In addition to the positive factors, some negative items stand out also. The main shortcoming was lack of supporting tools, other than the CPASCAL compiler. The need for automated tools was clearly evidenced in the delays experienced in handling of documents, diagrams, and source text files. Some tools which should be used in any comparable software project include:

--A document processor for all technical reports and design documents, with a facility for processing diagrams

--A module design and development system. The design part would maintain function specification, intermodule information, and status of modules. The development part would maintain libraries of listings and it would support separate compilation and testing of selected subsets of modules

--An interactive test facility would allow setting of variables, calling of submodules, output of intermediate results, and recording of test status information

Parts of all of the above tools have been demonstrated in other systems, but not in any single system both portable and compatible with CPASCAL. KSU has begun development of some parts of these tools for a CPASCAL system. During most of the work of the project, however, operations covered by these tools were done manually with the attendant delays and unreliability.

4.4 <u>Evaluation of Hardware (KSUBUS) Engineering</u>

The requirements to connect closely coupled heterogeneous minicomputers which support distributed data bases are as follows:

a. to move information from one computer memory to another

b. large amounts of it (64K bytes)

c. with a minimum of supervision

d. at a very high rate of speed

e. without effecting program execution in any of the computers significantly

f. from a multiplicity of computers

g. to a multiplicity of computers

h. simultaneously

i. allowing bypassing of broken links if possible

j. at very low cost

Briefly stated, these were the objectives for the MIMICS network hardware. Requirements (d) and (e) implied the use of Direct Memory Access (DMA) to the computers involved, with some kind of DMA-to-DMA cable connections. However, requirements (f) and (g) foretold trouble, since the number of DMAs which can be attached to any particular computer is quite low--often in the 1-2 range. First, design proposals considered a form of electronic selector switch to connect two DMAs with each other for the duration of a block transfer sequence, but requirement (h) could not be met without horrendous duplication of hardware.

Requirement (h) also implied LOTS OF CABLES, yet requirement (i) implied dynamic rerouting when a path of the network was disabled. Requirement (j) prohibits a multiplicity of cables anyway!

The basic requirements (a), (b), and (c) called for relatively autonomously operating hardware, and such was easy to provide. However, since the number and kind of computers which would be present in any particular network were not given in advance, very <u>modular</u> design was required so that changes could be accommodated without major redesign.

The final design developed into sets of autonomous functional units, called Data Movers, communicating with each other and with DMAs over a common short high-speed (5 MHz) bus called a KSUBUS. Each KSUBUS unit is attached to nearby computers via a single DMA. Each Data Mover is connected to a Data Mover on another KSUBUS via a medium-length (z 50 foot) multiwire cable. Each Data Mover contains sufficient hardware logic to transfer up to 65,536 2-byte units of information in a single block, given only the source and destination computers, the block length, and the beginning memory addresses for the information.

The prototype system was implemented using large handwired Douglas Electronic logic boards plugged into a spare Interdata expansion chassis and connected to several Interdata computers.

Following are comments in reference to requirements:

a. achieved

b. achieved - perhaps too well. No one at the present time normally transfers anywhere near that much information. However, the logic difference between 131,072 bytes and a smaller reasonable number, such as perhaps 512 bytes, is only five or six integrated circuit (IC) packages!

c. achieved

d. 5 MHz - 7.5 MHz should be possible with a crystal change if the ICs meet the manufacturer's "typical" specifications, on the average. Still higher speeds would require some redesign and a better fabrication method

e. achieved

f. achieved - depending on circumstances, from 1-15 computers can be connected to Data Movers on a single KSUBUS

g. achieved

h. achieved - to a resolution of 200 nanoseconds. The limiting factor is the speed of computer memories accessed by the DMAs

i. achieved - by allowing Data Movers on the same KSUBUS to communicate with each other without affecting the attached computers

j. achieved - using equipment already available at KSU. Slightly different equipment would have simplified several decisions and enhanced some of the results

CHAPTER 5

5.0 System Integration

The complexity of the network software necessitated a unique system integration procedure. The most common integration methodology is bottom-up. Thus, the lowest level modules are tested with all possible inputs. Successively higher levels of modules are tested assuming that lower levels of modules are error free. However, the activity of the line control level (see Figure 4.5) of the packet system is not predictable. (This is necessary for the bottom-up procedure.)

The following integration methodology was used [PBH 77]. Each message system monitor, class, and process was individually tested. The composition of message system components was then tested by adding one module at a time. This combination of modules included "simulated" user processes and a data mover monitor. It also included simulated line processes. This was necessary to achieve predictable behavior at this level. This implementation stage is shown in Figure 5.1.

Stage 2.1 of integration included a user envelope on a COBOL program. This is shown in Figure 5.2. It includes SYSQ and a data mover implemented in the local operating system. This structure was then carried out across two machines--an Interdata 8/32 and a 7/32. Figure 5.3 displays the message system off-loaded to the communication controller. This exact configuration was not tested due to time constraints. However, the testing of the KSUBUS integration was tested (described in Section 6.2). This configuration will be tested shortly.

Stage 2.2 is shown in Figure 5.4. This testing with the IBM 370 version of the message system [RAY 77] was carried out without incident since the upper level testing was extensive. Applications of these configurations are described in Section 6.

CHAPTER 6

6.0 Prototype Operation

6.1 Prototype Distributed DBMS

In order to demonstrate the viability of the message system software in a heterogeneous machine environment, a prototype distributed data base management system has been constructed. The distributed DBMS utilized TOTAL as its data base manager and the message system as its communication mechanism. The computer systems included in the data base network are the Interdata 8/32, Interdata 7/32 and IBM 370/158. The intermachine connections are as shown in Figure 6.1. The system has been designed so that all machines can function as either host or back-end processors. In the initial version of the prototype, the system will operate in a single user mode with fixed host and back-end processors. Expansion to a multi-user environment with the processors acting as bi-functional machines (performing both host and back-end functions) is planned in the near future.

The software structure of the distributed DBMS is illustrated by the information flow resulting from a data base request in the application program in Figures 6.2 and 6.3. The host interface (HINT) and back-end interface (BINT) programs serve to control and coordinate the communication between the application program and the data base manager. The host interface is called from the application program whenever it requires data base service. The HINT program packs the data base request into buffers and calls the host version of the message system to communicate with BINT on the back-end processor. When BINT receives the message, it unpacks the message buffers and calls DATBAS, the TOTAL data base manager. DATBAS returns its status and data information to BINT upon completion of the operation. BINT transmits the results of the data base command through HINT to the application program, which then proceeds in its execution sequence.

Both HINT and BINT are implemented in COBOL and, for purposes of the prototype, interface with COBOL application programs.

6.2 Prototype KSUBUS Operation

The testbed chosen to demonstrate the speed and utility of the KSUBUS is a distribution of operating systems functions across the bus. The objectives are to test the performance of the bus and to demonstrate such performance on an operational system. The performance is tested during the use of a file transfer protocol (FTP) which is implemented between a single user operating system (SOLO 0) and a line printer spooling (SPOOLER) subsystem.

The modified SOLO resides on one machine (Interdata 7/16), and the SPOOLER resides in another machine (Interdata 85) which controls the bus. The control information for the file transfer protocol is passed across the asynchronous control lines; and the file data can be moved either across the high-speed KSUBUS or the slow-speed asynchronous lines.

This topology is illustrated in Figure 6.3.

Since the critical element of a DDBMS is the connection, the data flow of the SOLO/SPOOLER system is intended to display the performance characteristics of a host/back-end system, respectively. That is, a user at a SOLO console requests a file transfer. It is then moved from the SOLO disk to the SPOOLER disk and back again to the SOLO disk. This same scenario is true of a user data base query request at a host terminal. The request is transmitted to the back-end and the data is returned. This type of data flow is illustrated in Figure 6.4.

The objective of the performance tests is to demonstrate the improvement in data transfer times when the KSUBUS is utilized instead of conventional slow-speed lines, such as those used in the prototype of the distributed data base management system of Section 6.1. The performance of the system in transferring data should be proportional to the speed of the transmission medium. In the prototype system, the system which moved data across the slow-speed lines achieved such efficiency. However, since the file protocol moved data from disk-to-disk, the system only ran at the speed of the disk even when data was moved across the bus. (If all data had been transferred memory-to-memory across the KSUBUS, only memory speeds would limit the system performance.) Since data bases are typically disk resident, this SOLO/SPOOLER system effectively displays the performance of a DBMS distributed across two machines connected via the KSUBUS. Furthermore, as faster devices are used for data base storage (such as charge-coupled devices) the same scenario will run faster--at the speed of the devices. Thus, since the KSUBUS runs at memory speeds, the performance of data base operations can be improved at the same rate as the technology of storage devices improves.

CHAPTER 7

7.0 Documentation

7.1 Overview

In the pursuance of the supported research, documentation of new concepts and theories and their prototype systems is provided in four (4) areas: data base management systems, hardware systems, software portability systems, and network control systems. Table 7.1 contains an enumeration of the documents in each area. The remainder of this chapter contains the abstracts of the reports produced within each of the four (4) areas.

Table 7.1

FUNCTIONALLY DISTRIBUTED
COMPUTER SYSTEMS

TECHNICAL REPORTS

PROGRESS REPORTS	SUMMARY REPORTS	PROJECT MANAGEMENT PLAN
MAY 1976	PART I	FEBRUARY 1976
DECEMBER 1976	PART II	

SOFTWARE UTILITY	HARDWARE SUPPORT	DATA BASE SYSTEMS	NETWORK CONTROL
MACRO LANG. (CS 77-12)	DESIGN OVERVIEW (CS 77-27)	BACK-END EVAL. (CS 76-08)	IMPL. DDBMS (CS 76-03)
SPASCAL (CS 76-18)	USER'S MANUAL (CS 78-01)	MINI-DDBMS (CS 76-11)	MIMICS DESIGN (CS 77-04)
CPASCAL (CS 76-17)	DESIGN DETAILS (CS 78-02)	BACK-END SIM N/ (CS 76-12)	CONTROL LINE (CS 77-15)
KSU CPASCAL (CS 76-16)		DDBMS SPEC. (CS 76-13)	MIMICS ARCH. (CS 78-04)
"PORTING" CPASCAL (CS 77-09)		MEM.MAN. DDBMS (CS 76-14)	MIMICS USERS (CS 78-03)
CPASCAL ARCH. (CS 77-12)		DBMS SURVEY (CS 77-01)	PS DOCUMENT. (CS 78-05)
		TRANSPARENT VIEW OF DDBMS (CS 76-22)	
		DEADLOCK DDBMS (CS 77-02)	
		ROLLBACK/RECOVERY (CS 77-05)	
		MULTI-HOST/BACKEND (CS 77-07)	
		DESIGN ASPECTS OF DDBMS (CS 77-08)	

Abstracts of Reports Produced in

Support of Army Grant DAAD-29-76-G-0108

CS 76-03 Maryanski and Wallentine. Implementation of a Distributed Data Base System. 18 pages. February 1976.

-ABSTRACT-

In this paper we present an overview of data base management systems (DBMS), the motivation for distributed data base systems (DDBMS), a set of possible network topologies served by the distribution, the mechanisms necessary to integrate (and communicate between) the DDBMS system elements when distributed across a nonhomogeneous network of minicomputers, and some implementation details on a prototype system. The current prototype distributes the DBMS and application program function across an IBM 370/158 and a (minicomputer) NOVA 2/10. In the near future, a third machine, the Interdata 85 minicomputer, will be added to the network. The DBMS used is a network system as specified by CODASYL. The emphasis in this paper will be on the problems posed by the heterogeneous machines and the intertask (processor) communication system which is utilized in the distribution of data, programs, and control.

CS 76-08 Fisher, Maryanski and Wallentine. Evaluation of Conversion to a Back-End Data Base Management System. 18 pages. March 1976. Published in the proceedings of the ACM Conference, 1976.

-ABSTRACT-

This paper presents a methodology for and an evaluation of the feasibility of converting a typical data processing system to a data base management system. This methodology is applied to a particular system. The data base management system under evaluation uses a back-end minicomputer to perform the data management functions. The evaluation is made in terms of changes in system resources, program requirements, and human factors. The results of this study provide considerable insight into the problem of conversion to a data base management system and suggest guidelines for the evaluation of any proposed data base conversions.

CS 76-11 Maryanski, Fisher, Wallentine, Calhoun and Sernowitz. A Minicomputer Based Distributed Data Base System. 20 pages. April 1976. Published in the proceedings of the NBS-IEEE Trends and Application Symposium: Micro and Mini Systems, May 1976.

-ABSTRACT-

This paper described a data base management system under development at Kansas State University, intended for use in a network composed primarily of minicomputers. The report presents a description of the computers forming the network and their intercomputer communication system. The data base management system is a network type as specified by CODASYL. An extension of a CODASYL-type DBMS to multicomputer configurations is presented and several DBMS network topologies are discussed. We then conclude with a discussion of a completely distributed data base network.

CS 76-12 Maryanski and Wallentine. A Simulation Model of a Back-End Data Base Management System. 24 pages. April 1976. Published in the proceedings of the 7th Annual Pittsburg Modeling and Simulation Conference, April 1976.

-ABSTRACT-

This paper presents a simulation model of a back-end data base management system (DBMS). The purpose of the model is twofold: to determine the effect of several configuration parameters on system performance in a back-end DBMS in general and to accurately describe a particular back-end DBMS implementation. The essential concepts of back-end data base management systems are described in this report. A discussion of the workings of an implementation of a back-end DBMS is also provided. GPSS has been used to model the back-end DBMS. Simulation studies are being conducted to study the effects on changes in various parameters on system performance. Results are given on the relationship between such performance factors as the number of DBMS tasks processed and CPU utilization versus the system parameters of levels of multiprogramming, task switch times, type of machine interconnection, and line speeds.

CS 76-13 Maryanski. Language Specification for a Distributed Data Base Management System. 76 pages. May 1976.

-ABSTRACT-

This document is a proposal for a distributed data base management system (DDBMS). It represents the first phase of the DDBMS design portion of Grant 108. It is very important to note that this document is a proposal and also that the next phase of the design is the development of the functional specifications of the DDBMS. Therefore, it is essential that all interested parties respond with any corrections, additions, deletions, suggestions, etc. by July 1, 1976.

As it can easily be observed from this report, the implementation of the complete DDBMS will be an enormous task. Estimates range from 7 to 20 up to 50 person years of effort. A natural course is to design a full scale system and proceed with the implementation in an incremental manner. The implementation of a minimal prototype should be achieved as soon as possible for purposes of feasibility studies, testing, and morale. Another important consideration is that based upon the current resource allocation to the data base portion of Grant 108, it is unlikely that the Special Features described in Chapter VIII can be included in the initial DDBMS design.

CS 76-14 Maryanski. Memory Management in a Distributed Data Base Management System. 48 pages. October 1976.

-ABSTRACT-

A memory management scheme which incorporates an additional level of memory into the traditional primary-secondary storage hierarchy is proposed for utilization in distributed data base management systems. In this scheme, the memory of the back-end processor is used as an additional memory buffer. An optimal three-level memory management algorithm is presented along with an analysis of its cost in terms of page replacement. The expected performance improvement over the optimal algorithm for a two-level memory system is determined. The performance benefits of the three-level memory management are applicable to most distributed processing systems.

CS 76-16 Neal, Anderson, Ratliff, and Wallentine. KSU Implementation of Concurrent PASCAL - A Reference Manual. 74 pages. December 1976.

-ABSTRACT-

This manual is intended to serve in the following ways:

1. As an overview to the implementation approach
2. As a reference manual for the SOLO user on the 8/32
3. As a reference manual for the Sequential PASCAL programmer using SOLO
4. As a configuration guide to the SOLO systems maintenance personnel

This manual contains a description of the implementation of Concurrent PASCAL as a task under OS-32/MT on an Interdata 8/32 computer. Further, it contains a simple introduction to using SOLO under OS-32/MT, a set of device assignments and completion codes, an overview of the SOLO console operation, a programmer's reference manual to the interface between Sequential and Concurrent PASCAL programs, and an introduction to the Sequential PASCAL program prefix. It contains the information on how to reconfigure the KERNEL of Concurrent PASCAL and the virtual disc of SOLO in terms of its dependence on OS-32/MT. Finally, the appendices include an annotated prefix, the SOLO utility manuals, a description of the compiler cross reference implementation, OS-32/MT utilities supporting the PASCAL system, and packaging information.

CS 76-17 Wallentine and McBride. Concurrent PASCAL - A Tutorial. 134 pages. December 1976.

-ABSTRACT-

Concurrent PASCAL was designed and implemented by Per Brinch Hansen as a language to use to implement operating systems. The definition of the language is contained in reference [PBH 75B]. An introductory example of its use is in reference [PBHA]. An excellent example of the utility of the language is the implementation of the SOLO operating system [PBH 76] as a Concurrent PASCAL program. This document contains a set of smaller (but complete) and more diverse applications of the language. The utility of Concurrent PASCAL is tested in applications such as priority scheduling of resources, message systems, the data base reader/writer problem data link control procedures, and network interprocess communication systems. Evaluations of several good and not-so-good language features are included.

CS 76-18 Hankley and Rawlinson. Sequential PASCAL Supplement for FORTRAN Programmers: A Primer of Slides. 161 pages. December 1976.

-ABSTRACT-

This report consists of pairs of slides which are designed to serve as an instructional aid to introduce programmers, who can read FORTRAN, to Sequential PASCAL as running at KSU. [Sequential PASCAL is a variant of PASCAL which was defined by P. Brinch Hansen and A. Hartman at California Institute of Technology. SPASCAL differs from Wirch's definition of PASCAL in both restrictions and extensions.] The slides can be used as handouts or transparencies for an intensive seminar on PASCAL, or they can be used for self-study. However, there is minimal (almost no) narrative, only lists of features and notes and sample programs. Typical FORTRAN programs are presented along with the corresponding Sequential PASCAL program. The examples are presented in a sequence designed to allow the programmer to quickly grasp the similarities and differences between the two languages. Differences are emphasized through the use of illustrations and warning statements. Programming examples are also used to introduce the user to Sequential PASCAL capabilities which cannot be duplicated in FORTRAN.

CS 76-19 Neal. An Architectural Base for Concurrent PASCAL. 126 pages. July 15, 1977.

-ABSTRACT-

The programming language Concurrent PASCAL in its design and implementation has exerted a substantial influence upon the fields of operating systems and concurrent programming. The work reported in this thesis extends that influence to the field of computer architecture by analyzing the model of concurrency which supports Concurrent PASCAL. As background to the architectural model, three implementations of Concurrent PASCAL are discussed, including a description of the process of transporting an implementation from one computer to another with its associated insights and problems. Details of the architectural base include discussions of the control and data models. The control model discussion centers around state transitions and scheduling. The data model presents a hardware stack mechanism for the execution of Concurrent PASCAL programs, which is also suitable for other block-structured languages within the framework of the concurrent processing.

CS 76-22 Maryanski, Fisher and Wallentine. A User-Transparent Mechanism for the Distribution of a CODASYL Data Base Management System. 36 pages. December 1976.

-ABSTRACT-

A software organization is presented to provide for data definition and manipulation in a distributed data base management system. With the mechanism for distributing the data base proposed here, the physical location of the data is transparent to the user program. A Device Media Control Language is specified for the assignment of control of and access to a data base area to a set of processors. Procedures for reassignment of the control and access functions as well as the transfer of data between processors are provided. The basic hardware and software requirements for a computer network capable of supporting a distributed data base management system are discussed along with a specification of the software required for a processor in a distributed data base network.

CS 77-1 Maryanski. A Survey of Developments in Distributed Data Base Management Systems. 36 pages. January 1977. To be published in IEEE Computer. February 1978.

-ABSTRACT-

Recently we have witnessed the advent of general purpose data base management systems and important advances in computer networks. The combination of the two technologies to produce distributed data base management systems should be the next significant step in commercial systems development. A completely generalized distributed data base management system would reside on a heterogeneous computer network with different data base systems available at various processors. Communication and data transfer would be possible between any nodes in the network. The realization of this goal is still several years in the furture. However, considerable progress in the area of distributed data base systems has been made in both academic and industrial environments.

This report described the principal problem areas in distributed data base management system development. Distributed data base systems share many design problems with both single machine data base systems and computing networks, as well as introducing several new dilemmas.

Recent research in these problem areas is presented to

provide a picture of the state of the art of distributed data base development. In addition, the current status of the data base industry with respect to distributed processing is evaluated by reporting the current projects and future plans of selected (anonymous) data base vendors.

CS 77-2 Maryanski. A Deadlock Prevention Algorithm for Distributed Data Base Management Systems. 27 pages. February 1977.

-ABSTRACT-

The problem of deadlock in distributed data base management is analyzed in terms of performance effects of potential deadlock handling schemes. The performance tradeoffs of deadlock detection and deadlock prevention for distributed data base management systems are compared. Since the run-time overhead in deadlock prevention is projected to be less than for deadlock detection, an algorithm for preventing deadlocks in distributed data base systems is developed. The critical information for the deadlock prevention algorithm is maintained in a shared record list. The shared record list contains all shared access records for a set of tasks. Shared records lists are maintained dynamically by the run-time system. A proof that the algorithm prevents deadlocks in a distributed data base management system is provided along with a comprehensive example.

CS 77-4 Wallentine, Hankley, Anderson, Calhoun and Maryanski. Overview of the Design of the MIMICS Network Architecture. 142 pages. December 30, 1976.

-ABSTRACT-

The basis for MIMICS (MIni- MIcroComputer System) is the utility of both mini- and microcomputers in the support of a distributed data base system. The goal of the research is development of a prototype MIMICS on heterogeneous computers. This report documents our approach to the design of MIMICS in the areas of--

1. mechanisms for accessing data in the network;
2. hardware interconnection facilities;
3. network interprocess (message) communication

system; and

4. implementation approach.

The structure of this report is first to give an overview of the MIMICS architecture. We then present the results of our research into design considerations in a distributed data base system. This is followed by an overview of the message system (network interprocess communication system) in MIMICS and details of the MIMICS hardware architecture which we have developed for large capacity computer-to-computer (memory-to-memory) data transfer. Finally, we present our approach to implementation. We discuss the structure of the implementation of the design, the properties of that structure, our approach to portability of systems, and some concepts of the system´s implementation language (Concurrent PASCAL).

CS 77-5 Maryanski and Fisher. Roll-back and Recovery in Distributed Data Base Systems. 19 pages. February 1977.

-ABSTRACT-

One of the major obstacles to the widespread development and utilization of distributed data base management systems is the lack of an efficient recovery technique. A methodology is presented here for recovery of distributed data bases. The central operation of the recovery technique is rollback of a data base application task on the processor which controls access to the data. The rollback procedure restores the data base to its original state prior to the execution of the application task and determines the set of applications tasks which may have been effected by that task. Tasks that have not operated upon data altered by tasks being rolled back are not affected by the procedure. The rollback procedure attempts to minimize the the time and space requirements for recovery.

CS 77-7 Maryanski. Performance of Multi-Processor Backend Data Base Systems. 17 pages. April 1977. Published in proceedings of the Conference on Information Science Systems, The John Hopkins University, Baltimore, Maryland, April 1977.

-ABSTRACT-

The results of a simulation study intended to determine the circumstances under which it is beneficial to operate a data base management system with a multi-processor backend are presented. The basic concept of backend data base management systems and multi-processor backend systems are provided as background material. The general structure of the simulation model which has been implemented in GPSS is outlined. The results of the study indicate that the amount of CPU activity required by the data base management system is a determining factor with respect to the need for a multi-processor backend.

CS 77-8 Fisher and Maryanski. Design Considerations in Distributed Data Base Management Systems. 20 pages. April 1977.

-ABSTRACT-

With the advent of Data Base Management Systems (DBMS) and associated facilities (data dictionaries, query languages, report writers, etc.), the task of data organization, management, and storage has been given to a select group of specialists. These specialists (the Data Base Administrators (DBA) provide the necessary control, logging, and access information and software to the program. Such activity relieves the programmers of this overhead function allowing them to concentrate on the necessary manipulations.

This paper focuses on some alternatives with respect to a DBMS in terms of a centralized versus decentralized environment. The first section of this paper deals with the concepts and tradeoffs involved in considering the two environments. The second section of the paper then deals with problems which are encountered in a distributed data base management system. These problems include deadlock, rollback and recovery, data conversion, redundancy, and communication and operating system requirements for effective distribution.

CS 77-9 Neal and Wallentine. Experience in Porting Concurrent PASCAL. June 1977.

-ABSTRACT-

The process of transporting Brinch Hansen's implementation of Concurrent PASCAL to another minicomputer is described. Applicable porting strategies are discussed with emphasis on the design decisions made for a specific transportation. Important design decisions include the use of a virtual code interpreter and implementation in an operating system environment. The problems of this transportation are illustrated with accompanying suggestions for a more portable system.

CS 77-12 Fisher, Hankley and Maryanski. Porting Software to Multiple Mini's: A DBMS Case Study. 23 pages. December 1976.

-ABSTRACT-

As minicomputer systems gain wider acceptance, the objective of developing portable minicomputer software becomes more compelling. Motivated by the task of making a data base management system available on different minicomputer configurations, this paper addresses minicomputer software portability. The need for designing portable software is emphasized and guidelines for such designs are developed. Alternative options are presented for the case study of synthesizing a portable data base management system, and the particular method selected is discussed in detail.

CS 77-15 Rehme and Wallentine. MIMICS (asynchronous) Control Line Protocol. 103 pages. December 1977.

-ABSTRACT-

This report contains a description of the design and implementation of an asynchronous control line driver in the MIMICS network. The driver handles the functions necessary for the transmitting and receiving of control information between computers within a cluster of the network. In the report we give a brief description of the MIMICS network and how the driver is used in that network. We then describe the use of asynchronous lines for communication, why they were chosen for this particular project, and how they are programmed on the Interdata 85 and the Interdata 7/16. It also tells how the computers were wired together to insure that the interface boards could detect abnormal conditions of the line. The implementation of the driver on the Interdata machines using assembler language and PASCAL is then presented, followed by a summary of the work completed and some extensions to conclude the report.

CS 77-27 Goodell. Control Computer Local Driver Routines in a Functionally Distributed Data Base Management System. 1977.

-ABSTRACT-

The Functionally Distributed Data Base Management System links computers in one geographic location together into a cluster and then forms a network with remote (distant) clusters, providing a system where each machine in the network operates in a specific computer area and each data base in the system is managed by one specific machine. To control this network, a second, smaller computer (ultimately a microcomputer) is allied with each main or host computer in the system. This control computer receives and issues instructions from and to the host computer or other control computers to arrange the movement of data from the memory of one computer to the memory of any other computer in the network. This project described local driver routines which direct the handwired logic of local data moving mechanisms. Included are detailed descriptions of the actions required by each request and an explanation of the software-hardware relationship.

CS 78-01 Calhoun. Functional Description of the MIMICS KSUBUS and Associated Hardware.

-ABSTRACT-

This document describes the overall functional specifications and network architecture of the high-speed KSUBUS and all of the associated hardware units: Direct Memory Access, Data Mover (Transmitter, Receiver, and Transmitter/Receiver), Remote Direct Memory Access, and Universal Logic Interface. A description of each of the buses comprising the KSUBUS is included. Data transfer mechanisms and transfer rates are discussed.

An appendix derives the maximum data transfer on a KSUBUS.

CS 78-02 Calhoun. Detailed Description of the MIMICS KSUBUS Hardware.

-ABSTRACT-

This document gives a detailed description of the KSUBUS and each of its associated hardware units: Direct Memory Access, Data Mover (Transmitter, Receiver, and Transmitter/Receiver), Remote Direct Memory Access, and Universal Logic Interface. The Digital Design System is also described.

Each module of every hardware unit designed at Kansas State Unversity is described in detail.

CS 78-03 Hankley, Wallentine, et al. MIMICS Message System: Introduction and User´s Guide.

-ABSTRACT-

MIMICS (MIni- MIcroComputer System) is a model for a network of computers, possibly large machines, but normally minicomputers. Communications functions within the network are designed to be off-loaded into microcomputer communications controllers. The MIMICS network was designed to be able to support quite arbitrary configurations of distributed data bases. The MIMICS structure was intended to eventually incorporate a distributed network operating system (DNOS); however, the prototype design and implementation includes just a network message handling system. The message system (MS) consists of distributed control software and hardware which allows cooperating user tasks, anywhere in the network, to send and receive large blocks of text data using very simple operations and protocols. This report presents a guide for users of the message system. It is written assuming a "typical" machine and using PASCAL-like notations to describe data structures and parameters. Supplemental guides are available for users on the Interdata 8/32 [DNN 76B] and IBM S/370 [RAY 77]. A companion report [HAW 78] provides a guide to the design and CPASCAL implementation of the message system. The summary report for the project presents an overview of all of these related documents [WHC 77]. The guide is organized in four parts. Section two presents the major features of the MIMICS design. These help the user to understand the system, but they are not absolutely necessary to the most naive users. Section three presents the actual naming conventions and explanations of how the message system is to be used. Section four tabulates the specific calling forms needed to use the message system. Section five presents scenarios which illustrate different kinds of use of the

message system.

CS 78-04 Hankley, Wallentine, et al. MIMICS Message System: Architecture and Implementation.

-ABSTRACT-

This report contains the functional and implementation documentation for the MIMICS message system. An overview of the structure (in Concurrent PASCAL) and data flow is presented. Functional specifications for each module in the system are given. This is followed by detailed algorithmic specifications. Concurrent PASCAL code for the system is attached as an appendix.

CS 78-05 Ratliff. Implementation of the MIMICS Packet Switch.

-ABSTRACT-

The MIMICS (MIni- MIcroComputer System) is a general purpose network system developed at Kansas State Unviersity. The structure of MIMICS is such that high-speed data transfers among members of a "cluster" are controlled by the cluster system, while transfers between distant machines utilize common carrier hardware controlled by the "packet switch." This report documents the implementation of this packet switch. Central to the understanding of the packet switch implementation are the concepts of communications across a logical line, not physical, and the concept of a logical line window for easy recovery and flow control across all physical lines which make up a logical line. The packet switch is responsible for buffering incoming and outgoing packets, routing packets based on destination information in the packet, and scheduling packets based on their priority. Enforcement of flow control and buffer allocation insures that no one task can monopolize all of the buffers and no one class of packets can completely preempt transmission of other classes of packets. The type of common carrier hardware used, line discipline, packet format and protocol used are extremely well isolated and rely largely on capabilities supplied by the operating system on which the MIMICS system is to run.

CHAPTER 8

8.0 Project Summary

The prototype message system software was written in Concurrent PASCAL. Its documentation consists of two overview documents [CS 76-03] [CS 76-11], a design document [CS 77-04], a user's manual [CS 78-03], an architecture document [CS 78-04], and two functional specification reports [CS 78-05] [CS77-15]. A 9-track tape contains the software in virtual code which can be run on an Interdata 8/32 or 7/32 under OS-32/MT or on a PDP-11 with no operating system support.

Since the use (adapability) of Concurrent PASCAL is enhanced by the use of the SOLO operating system, a 9-track tape contains the SOLO system, the "ported" 8/32 system, the "ported" 16-bit compilers, and spooling subsystem. A tutorial on Sequential PASCAL and one on Concurrent PASCAL are included. Two documents on the "porting" of Concurrent PASCAL are also included.

The construction specifications for the KSUBUS prototype (and all of its associated interfaces) are presented in reference [CS 78-02]. A user's manual [CS 77-27] and a basic architecture guide [CS 78-01] are included.

A 9-track tape is provided which contains the prototype distributed data base system. Eleven (11) technical reports have been published which isolate performance characteristics and mechanisms to achieve a distributed data base system.

In summary, twenty-six reports (and three progress reports) were produced within the scope of this grant. A prototype network message system was developed which consists of 5000 lines of Concurrent PASCAL code which generates 50K bytes of machine code. A prototype high-speed bus was developed which consists of the control interface, the bus, and three local data movers. The software is available on magnetic tape.

Appendix A

Articles and Publications

Technical Reports, KSU

Number	Page	Abstract Authors and Titles
CS 76-03	42	Maryanski and Wallentine. Implementation of a Distributed Data Base System. 18 pages. February 1976.
CS 76-08	42	Fisher, Maryanski and Wallentine. Evaluation of Conversion to a Back-End Data Management System. 18 pages. Published in the proceedings of ACM National Conference. October 1976.
CS 76-11	43	Maryanski, Fisher, Wallentine, Calhoun and Sernowitz. A Minicomputer Based Distributed Data Base System. 20 pages. April 1976. Published in the proceedings of the NBS-IEEE Trends and Application Symposium: Micro and Mini Systems. May 1976.
CS 76-12	43	Maryanski and Wallentine. A Simulation Model of a Back-End Data Base Management System. 24 pages. April, 1976. Published in the proceedings of the Seventh Annual Pittsburg Modeling and Simulation Conference. April 1976.
CS 76-13	44	Maryanski. Language Specification for a Distributed Data Base Management System. 76 pages. May 1976.
CS 76-14	44	Maryanski. Memory Management in a Distributed Data Base Management System. 48 pages. October 1976.
CS 76-16	45	Neal, Anderson, Ratliff and Wallentine. KSU Implementation of Concurrent PASCAL - A Reference Manual. 69 pages. December 1976.
CS 76-17	45	Wallentine and McBride. Concurrent PASCAL - A Tutorial. 129 pages.

		December 1976.
CS 76-18	46	Hankley and Rawlinson. Sequential PASCAL Supplement for FORTRAN Programmers: A Primer of Slides. 145 pages. December 1976.
CS 76-19	46	Neal. An Architectural Base for Concurrent PASCAL. 126 pages. July 15, 1977.
CS 76-22	47	Maryanski, Fisher and Wallentine. A User-Transparent Mechanism for the Distribution of a CODASYL Data Base Management System. 34 pages. December 1976.
CS 77-1	47	Maryanski. A Survey of Developments in Distributed Data Base Management Systems. February 1977.
CS 77-2	48	Maryanski. A Deadlock Prevention Algorithm for Distributed Data Base Management Systems. February 1977.
CS 77-4	48	Wallentine, Hankley, Anderson, Calhoun and Maryanski. Progress Report on Functionally Distributed Computer Systems Development. 147 pages. December 1976.
CS 77-5	49	Maryanski and Fisher. Roll-back and Recovery in Distributed Data Base Systems. 19 pages. February 1977.
CS 77-7	49	Maryanski. Performance of Multi-processor Backend Data Base Systems. 15 pages. April 1977.
CS 77-8	50	Fisher and Maryanski. Design Considerations in Distributed Data Base Management Systems. 19 pages. April 1977.
CS 77-9	50	Neal and Wallentine. Experience in Porting Concurrent PASCAL. June 1977.
CS 77-12	51	Fisher, Hankley, and Maryanski. Porting Software to Multiple Mini's: A DBMS Case Study. 23 pages. December 1976.
CS 77-15	51	Rehme and Wallentine. MIMICS (asynchronous) Control Line Protocol. 103 pages. December 1977.

CS 77-27	52	Goodell. <u>Control Computer Local Driver Routines in a Functionally Distributed Data Base Management System</u>. 1977.
CS 78-01	52	Calhoun. <u>Functional Description of the MIMICS KSUBUS and Associated</u> Hardware.
CS 78-02	53	Calhoun. <u>Detailed Description of the MIMICS KSUBUS Hardware</u>.
CS 78-03	53	Hankley, Wallentine, et al. <u>MIMICS Message System: Introduction and User´s Guide</u>.
CS 78-04	54	Hankley, Wallentine, et al. <u>MIMICS Message System: Architecture and Implementation</u>.
CS 78-05	54	Ratliff. <u>Implementation of the MIMICS Packet Switch</u>.

Appendix B

Reports

Date	Subject
March 24, 1976	Report of Monthly Review
April 21, 1976	Report of Monthly Review
May 20, 1976	Progress Report
September 1, 1976	Report of In-process Review
July 1, 1976	Progress Report to ARO
January 27, 1977	Progress Report to ARO
September 15, 1977	Report of In-process Review
January 30, 1978	Final Technical Report to ARO

Appendix C

Table 1 Vocabulary

In discussing the MIMICS network concepts and implementation, it is essential to establish certain base vocabulary. Several of these key words are explained in the list which follows. Each word has been graded using the following scheme:

(1) ------ means word is essential for network users

(2) ------ means word is needed for discussion of network concepts

(3) ------ word is related to network implementation

(1) network - an interconnected set of computers.

(1) MIMICS - a network designed to be implemented using MIni- and MIcroComputers, but also with larger machines in the network; developed at KSU under support from the U.S. Army Computer Systems Command.

(1) connected - the network hardware is said to be connected if it is possible for communication to flow from any one machine to any other machine in the network, either directly or indirectly via intermedidate machines; MIMICS is intended to be connected.

- two user tasks are said to be connected if they have mutually established a "logical connection" by appropriate matching MS_CONNECT calls; these tasks may then communicate using MS_SEND and MS_RCV calls.

(1) user task - an application task in one of the network host machines that communicates to some other user

task, likely in a different machine, using the message system.

(1) <u>message system</u> - that software/hardware part of MIMICS that supports network communciation by user tasks; basic message system commands are CONNECT, DISCONNECT, SEND, and RCV (receive); basic message system functions are routing of messages, packetizing messages for remote transmissions, buffering of packets, handling of line protocol for packets and messages, and reconstruction of packetized messages.

(2) <u>remote</u> - two machines (or user tasks) are remote if communciation between them must travel over low-speed telecommunciations lines (e.g., 2400 baud, synchronous); messages between remote tasks are packetized by the message system, i.e., broken into packets for transmission and reconstructed at the receiving machine; opposite of local.

(2) <u>local</u> - two user tasks are local if either (i) they are in the same machine or (ii) they are in machines connected by high-speed "data movers" (e.g., 2 million bits per sec); messages between local tasks are <u>not</u> packetized, then are sent as a block, memory-to-memory using the data movers; each group of local machines is called a cluster; opposite of remote.

(1) <u>host</u> - any computer in the network with user tasks in it; warning--this differs from usual data base terminology as in a distributed data base application; both the front-end and back-end computers would be called network hosts; in MIMICS, hosts may be either

minicomputers or maxicomputers.

(2) off-loading - the removing of some operating system or language support functions from a host machine to an allied dedicated processor; the motivation for this is that the off-loaded functions can execute truly concurrently (i.e., simultaneously) with tasks in the host, thus greatly improving the performance of the host; in MIMICS, the message system is typically off-loaded into a communications control (CC) microprocessor; in the 370 architecture, the I/O functions are off-loaded to special channel control processors.

(2) CC - communication controller; a microprocessor used in MIMICS for off-loading the message system from a host machine.

(1) message - basic unit of network communication; copied by the message system from address space of a sender user task into agreed upon place in address space of a receiver user task; in MIMICS, messages may have two components, (i) a command part (up to 128 bytes of data) and (ii) a data part (up to 64K bytes), but either (not both) of the parts may be null.

(2) routing - selection of the path between two host machines over which communication will flow--hence, the selection of (i) which intermediate machines, if any, are part of the path and (ii) which actual communication line, in case there is more than one, to use between any two directly connected machines; in MIMICS, each message system instance has a route table

with entries <name_of_another_machine_in_the_network: line_route_to_next_machine_in_the_path> where the line route number is a logic line, so that all physical lines to an adjacent machine are used interchangeably.

(3) logical line - a group of parallel physical communications lines which directly connect two adjacent computers, where the actual physical lines are used interchangeably; warning--this means that packets can flow "out-of-sequence," although user tasks never observe this phenomenon.

(3) KSUBUS - a special multiplexed hardware bus, designed by M. Calhoun at KSU, to form a memory-speed connection between a CC, one or two hosts which are on the bus, an XR-data mover, and X- and R-data mover pairs which connect to other KSUBUS's in the same cluster.

(1) cluster - in MIMICS, a group of network machines that are all interconnected by high-speed data movers; the data parts of messages move at memory speeds from the sender task to the receiver task.

(3) c-node - a cluster-node; the group of one or two hosts which are connected to the same KSUBUS; messages can move memory-to-memory within a c-node without accompanying cluster protocol; warning--in conventional network terminology, any machine in the network would be called a node, but that is different from the c-node concept.

(3) data mover - a special "Autonomous Functional hardware Unit," designed at KSU, to work in conjunction with other matching units to move data blocks

memory-to-memory at memory speeds between machines in the same cluster; XR-, X- and R-data movers; a data mover can be enabled only by the CC on the same KSUBUS as the data mover.

(3) XR - data mover - device which copies a block of data from one area to another within machines on the same KSUBUS, e.g., host-to-host or host-to-CC or CC-to-host.

(3) X-data mover - device which "transmits" a block of data to an R-unit on a connected KSUBUS, where the source of the data is either (i) memory of a machine on the same KSUBUS with the X-unit (called X-mitting) or (ii) an R-unit on the same KSUBUS as the X-unit (which is called forwarding of the data).

(3) R-data mover - device which receives data from an X-unit on a connected KSUBUS and "moves" the data to either (i) memory of a machine on the same KSUBUS as the R-unit (called receiving) or (ii) to an X-unit on the same KSUBUS (called forwarding).

(3) packet - a basic unit for communication over a low-speed line; in MIMICS, the packets have the following components:

beginning_part = 6-SYNs - DLE-STX

packet_flow_control (4 bytes) =

RC---return control character

RN---return sequence character

N----out sequence character

TL---text_length character

message_flow_control (12 bytes =

SEQ------packet sequence number (2 bytes)

T--------type of packet character

ID-------message id character

TO_ID----4 bytes

FROM_ID--4 bytes

packet_text (0 to 128 bytes of data character plus transparency characters as required plus extra SYN characters as needed)

check_sum_part (2 bytes)

end_part = DLE-ETX

This comprises normally up to 156 characters, and most likely several more, to transmit data text of up to 128 bytes, so that the effective line baud rate is less than the nominal baud rate. Transmission errors and subsequent retransmission reduce the effective line baud rate even further.

(2) buffering - mechanism for providing space (buffers, actually "empty" buffers) and temporarily storing information (also called buffers, or full buffers), so that the related steps of storing and removing buffers (actual contents of the buffers) can proceed asynchronously, with the cumulative number of stores at all times ahead of the cumulative number of removals.

Buffers in MIMICS include:

(2) SYSQUE - buffer between user tasks and message system; buffers requests to message system and responses back to user tasks.

(2) protocol - an agreed upon form and sequence for exchange of control information and data between

processes to achieve a synchronized communication, i.e., so that the information is correctly conveyed and both processes know it; there are several sets of protocol in MIMICS, including:

(1) SYSQUE protocol - protocol for both user tasks and message system to both send and receive SCBs, which are control blocks used to implement passing of parameter information for message requests and responses.

(1) Message system - set of parameters lists for message system requests together with rules for acceptable user task behavior.

(3) Synchronous line - rules of sequencing for exchanging packets between remote line drivers.

(3) CC protocol - actually two sets of protocols;

(i) rules for exchanging packets between cluster - CCs (same as synchronous line protocol) and

(ii) rules for controlling the data mover's copying of data blocks within the cluster.

(3) PASCAL - a programming language designed by N. Wirth which promotes correct programs because (i) it promotes structured programs (both flow of control and data structures) and (ii) it enforces numerous compile time checks not normally supported in other programming languages (thus minimizing run-time errors), and (iii) it allows code to be written in a very easily readable

form.

(3) PASCAL - at the same time, a restriction of PASCAL to enforce simple programs and an extension of PASCAL to support a well-structured mechanism for concurrent programs using monitors; developed by P. Brinch Hansen; ported to KSU for use in implementing a readable and correct prototype of the message system.

(3) monitor - a concept introduced by C. Hoare for structured programming of concurrent processes; the monitor consists of (i) a group of shared data structures, (ii) a set of procedures (monitor entry points) which operate on the shared data, (iii) and initial state for the shared data, and (iv) the convention that only one process may execute "in" the monitor at any one time, so that the programmer does not have to worry about difficulties of multiple processes writing to the shared data at the same time; monitors are implemented in CPASCAL; monitors in the MIMICS implementation include:

- SYSQUE - monitor of SCBs for message system requests and responses
- packet_buffer - monitor of packets to be sent or just received
- MESS_TABLE - monitor of active and queued SEND and RCV requests
- CONNECT_TABLE - monitor of user task connection status information.
- logical_line_window - monitor of packets actively being transmitted, received, or acknowledged

over low-speed lines; one for each logical line

- event_control - monitors to control a process which has to await availability of data in either of two (or more) other monitors, since a process in CPASCAL can normally wait on only one monitor
- cluster_monitor - monitor of request and responses for activation of the data movers

(1) <u>NRC</u> - Network Resource Controller; a network operating system module needed to interface user tasks between the local operating system in the host machine and the network operating system; one for each host machine; functions of this NRC include:

- supplying network names to each user task
- initiating tasks in a host upon request from the NRC in some other host (based upon requests from user programs)
- disconnecting user tasks from the message system when the task terminates without the normally expected disconnect step

(1) <u>local operating system</u> - the regular operating system in any single host machine.

(1) <u>network operating system</u> - the collection of all operating software in all network machines including all NRCs, all message system instances, all SYSQUEs, etc.

(2) <u>user envelope</u> - interface software to translate message system calls in user programs to appropriate usage of

the SYSQUE; in particular, the user envelope will need to supply specific network names for all communications requests.

(1) network names (c.m.t.p.) - all communications in MIMICS are directed using a network-wide naming convention consisting of four bytes:

c = cluster character

m = machine

t = unique task identifying character, within machine c.m.

p = port character: the port character effectively identifies a communication subname so that one task may carry one network communication using two different ports and keep messages to each port separate.

c, m, and t names for a task can be established by interrogating the NRC.

(1) local names - within a host, tasks will be identified by names assigned by the local operating system; these are not network names; warning--it is necessary to translate between local names and network names in order to interface user tasks to both the local operating system and the network operating system.

(2) back-end - typically refers to a host computer executing only a data base management function; sometimes refers to the function inside a partition in a host which executes application programs in other partitions.

(3) packet buffer monitor - buffers packets to be sent over

low-speed lines and received from a low-speed line.

(3) <u>line drivers</u> - buffers the packets as they are actually being transmitted or received over a low speed line.

(1) <u>message table</u> - buffers SEND and RCV requests that have been accepted by the message system but not yet completed.

Appendix D

References

[CAH 74] Hoare, C.A.R. Monitors: An Operating System Structuring Concept. Communications of ACM, Vol. 17, No. 10, October 1974.

[CEK 74] Cerf and Kahn. "A Protocol for Packet Network Inter-communication." IEEE Transactions on Communications, Vol. Com-22, No. 5, May 1974.

[DM 78] Mounday, D. Porting CPASCAL to the NCR 8250. Kansas State University M.S. Report, 1978.

[DNN 76A] Neal, D.N. An Architectural Base for Concurrent PASCAL. (M.S. Thesis) KSU Department of Computer Science, Technical Report CS 76-19, November 1976.

[DNN 76B] Neal, D.N., Anderson, G., Ratliff, J. and Wallentine, V. KSU Implementation of Concurrent PASCAL - A Reference Manual. KSU Department of Computer Science, Technical Report CS 76-16.

[DNN 77] Neal, D. and Wallentine, V. Experience in Porting Concurrent PASCAL. KSU Department of Computer Science, Technical Report CS 77-9, June 1977.

[FJM 76] Maryanski, F.J. Design Considerations for a Distributed Data Base Management System. KSU Department of Computer Science, Technical Report CS 76-14, September 1976.

[GVB 75] Bochmann, G.V. Logical Verification and Implementation of Protocols. Fourth Data Communications Symposium, October 1975.

[INT A] INTERDATA INC. 16 Bit Series Reference Manual. Pub. No. 29-398R03.

[INT B] INTERDATA INC. MODEL 8/32 Processor User's Manual. Pub. No. 29-428.

[INT C] INTERDATA INC. OS-32/MT Program Reference Manual. Pub. No. B29-390R02.

[JHH 76] Howard, J.H. Signaling in Monitors. Proceedings of 2nd International Conference on Software Engineering (ACM/IEEE/NBS), (IEEE Cat. No. 76 CH1125-4 C), October 1976.

[J&W 74] Jensen, K. and Wirth, N. PaSCAL - User Manual and Report in Lecture Notes in Computer Science. No. 18, Springer Verlag, 1974.

[LOH 77] Lohr, Klaus-Peter. Beyond Concurrent PASCAL. Proc. of 6th ACM Symposium on Operating Systems Principles, November 1977.

[MAC 78A] Calhoun. M.A. Functional Description of the MIMICS KSUBUS and Associated Hardware. KSU Department of Computer Science, Technical Report CS 78-01.

[MAC 78B] Calhoun. M.A. Detailed Description of the MIMICS KSUBUS Hardware. KSU Department of Computer Science, Technical Report CS 78-02.

[MB 76] Ball, M. Personal Communication. Naval Underseas Laboratory, San Diego, CA.

[NW 71] Wirth, N. The Programming Language PASCAL. ACTA Informatica, Vol. 1, No. 1, 1971, pp. 35-63.

[PBH 73] Brinch Hansen, P. Concurrent Programming Concepts. ACM Computing Surveys, Vol. 5, No. 4, December, 1973.

[PBH 75A] Brinch Hansen, P. The Programming Language Concurrent PASCAL. IEEE Transactions on Software Engineering, Vol. 1, No. 2, June 1975, pp. 199-207.

[PBH 75B] Brinch Hansen, P. Concurrent PASCAL Report. Information Science, California Institute of Technology, June 1975.

[PBH 76] Brinch Hansen, P. The SOLO Operating System. Software Practice and Experience, Vol. 6, No. 2, April-June 1976, pp. 141-206.

[PBH 77] Brinch Hansen, P. The Architecture of Concurrent Programs. Prentice-Hall, 1977.

[RAY 78] Young, R. IBM System/370 Implementation of the MIMICS Network Message System. KSU Department of Computer Science, Technical Report CS 78-06, May 1977.

[REW 77] Rehme, E. and Wallentine, V. MIMICS (Asynchronous) Control Line Protocol. KSU Department of Computer Science, Technical Report CS 77-15, December 1977.

[SIB 77] Silberschantz et al. Extending Concurrent PASCAL to Allow Dynamic Resource Management. IEEE Transations on Software Engineering, Vol. SE-3, No. 3, May 1977.

[SIL 77] Silberschantz et al. On the Input/Output Mechanism of Concurrent PASCAL. Proceedings of the IEEE COMPSAC 77 Conference, November 1977.

[WAL 72] Walden, D. A System for Interprocess Communication in a Resource Sharing Computer Network. Communications of the ACM, Vol. 15, No. 7, April 1972.

[WJH 76] Hankley, W.J. et al. Sequential PASCAL Supplement (for FORTRAN Programmers). KSU Department of Computer Science, Technical Report CS 76-18, November 1976.

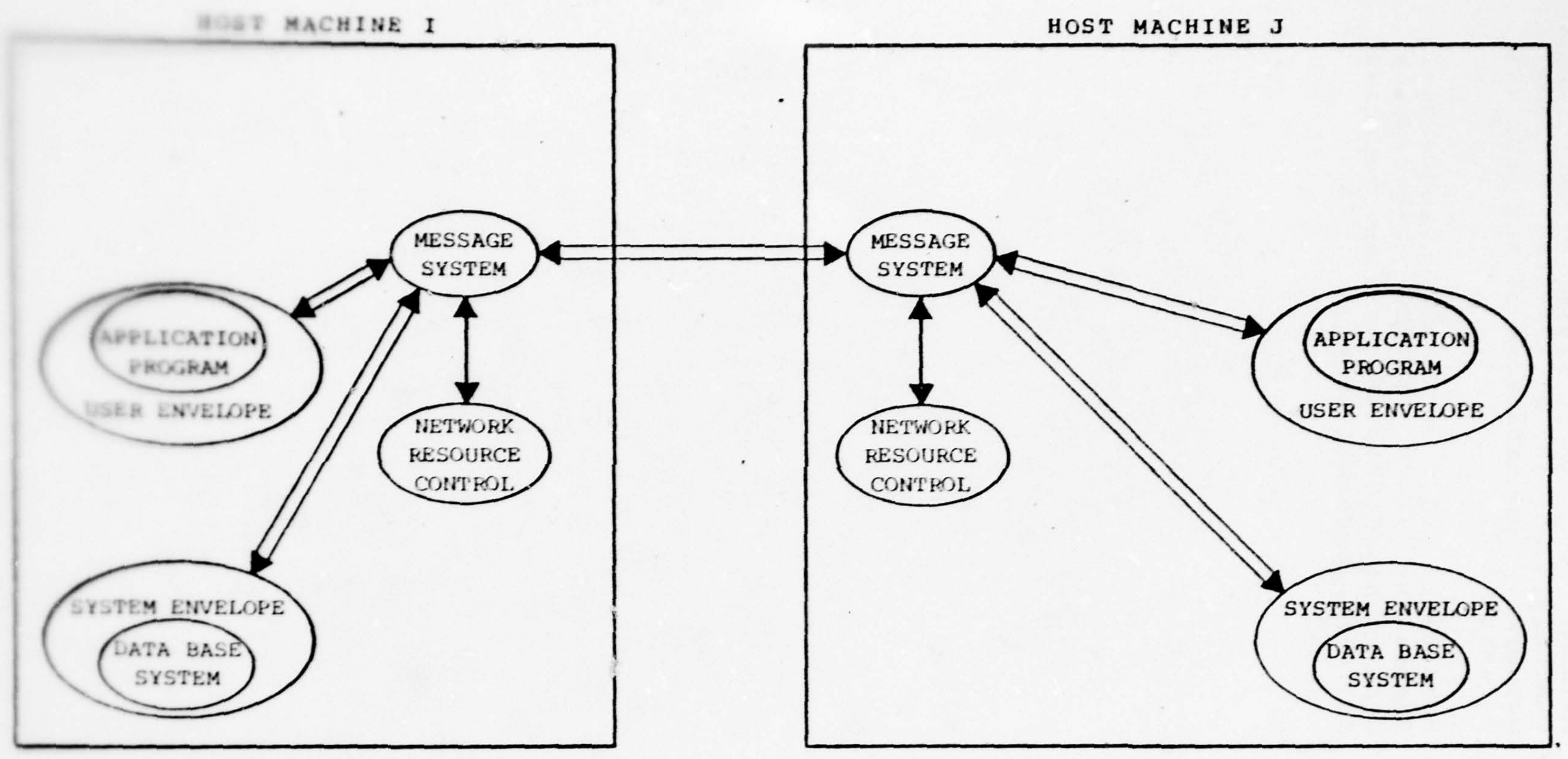


Figure 4.1A

Software Distribution in Host

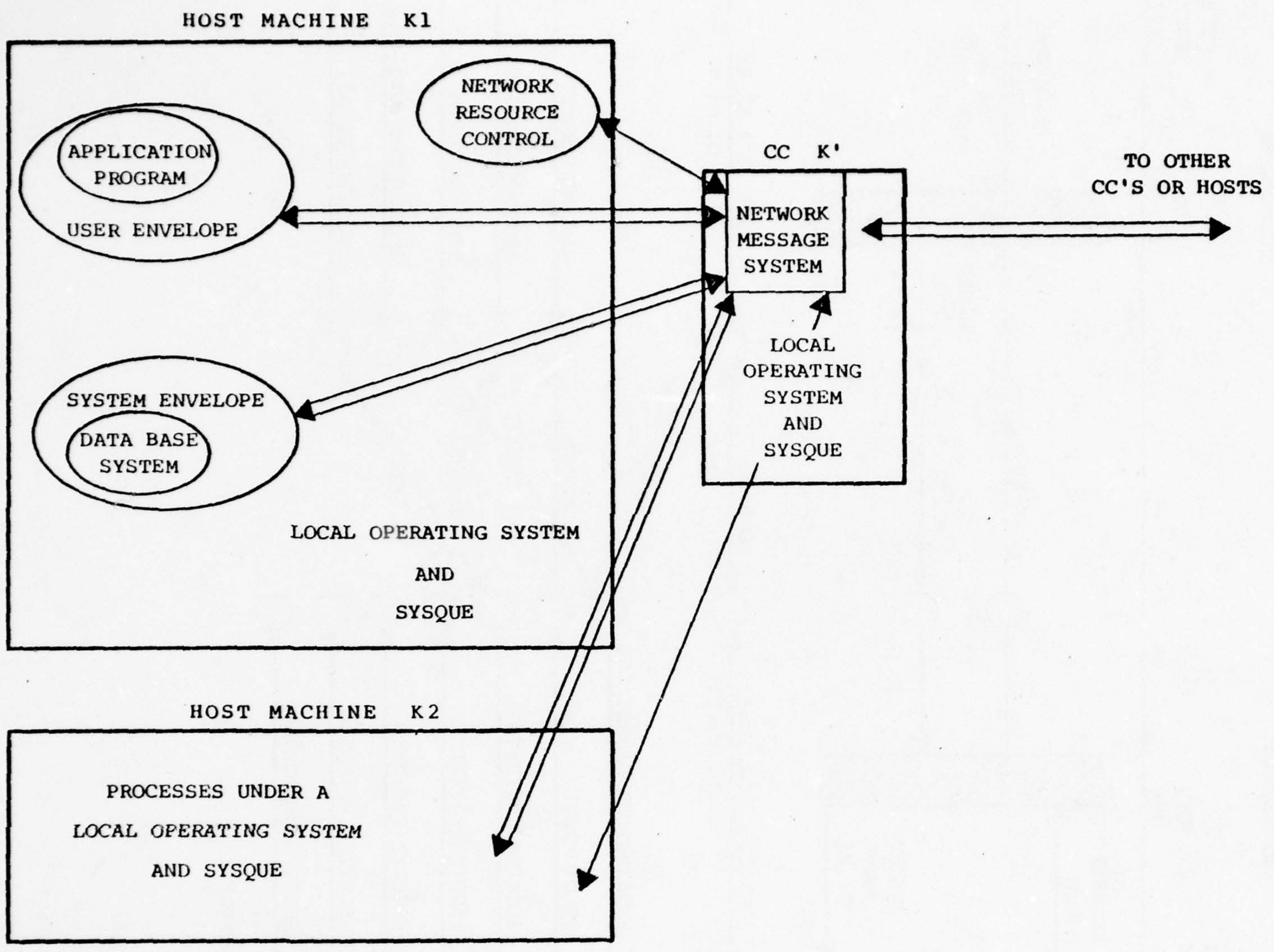


Figure 4.1B

Software Distribution Across Host/CC

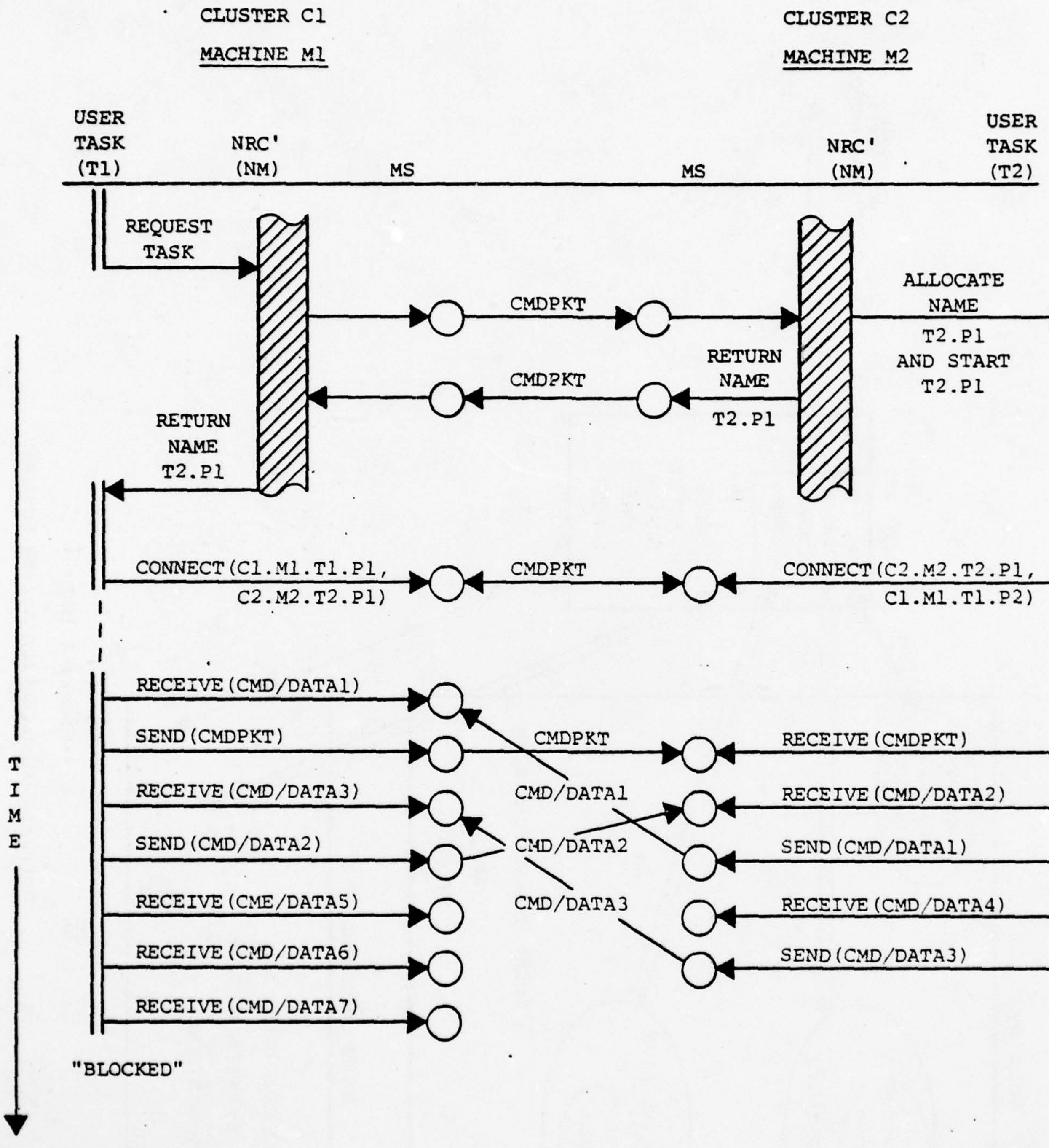


Figure 4.2

Connection/Synchronization of

Type 2 Communication

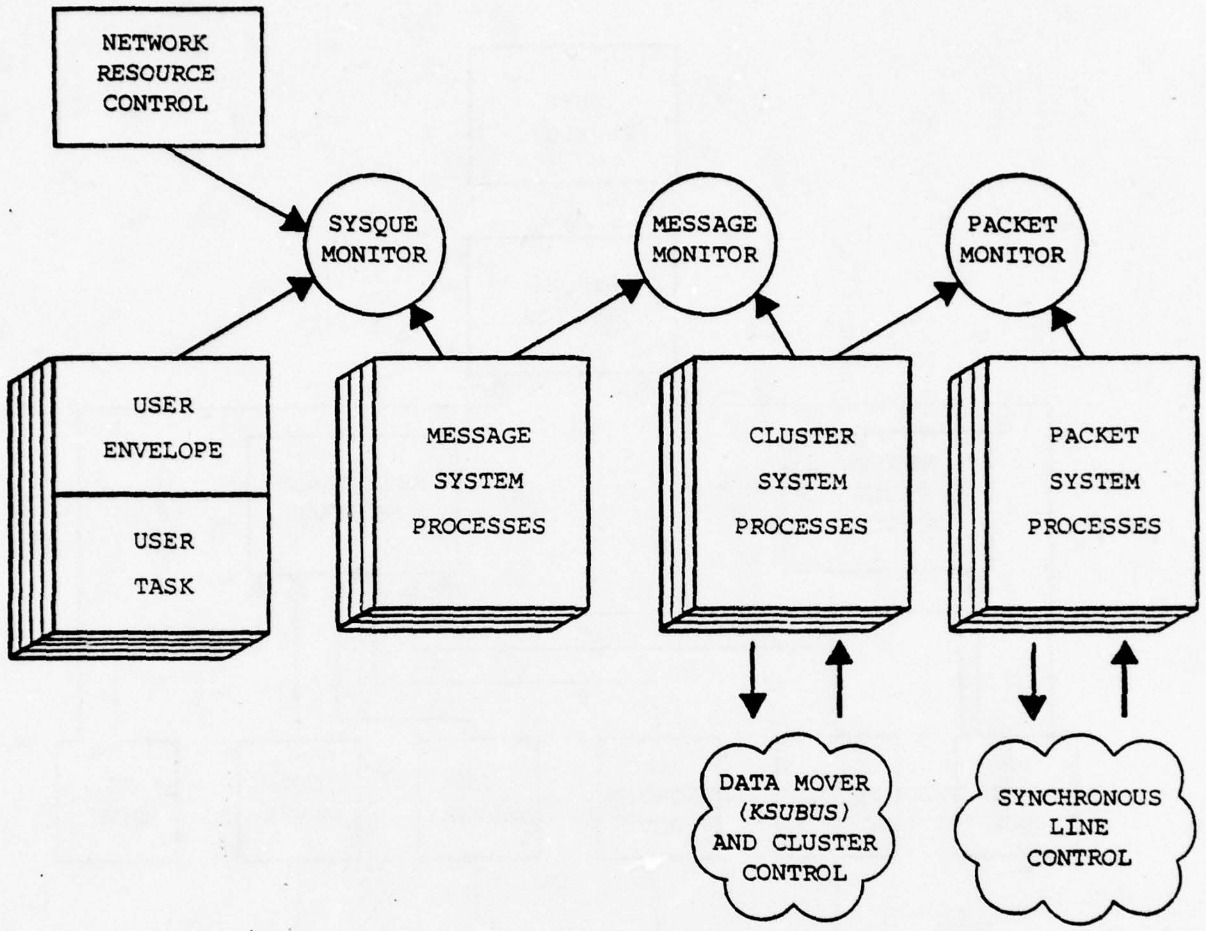


Figure 4.3

Network (IPC) Network System

Implementation Structure

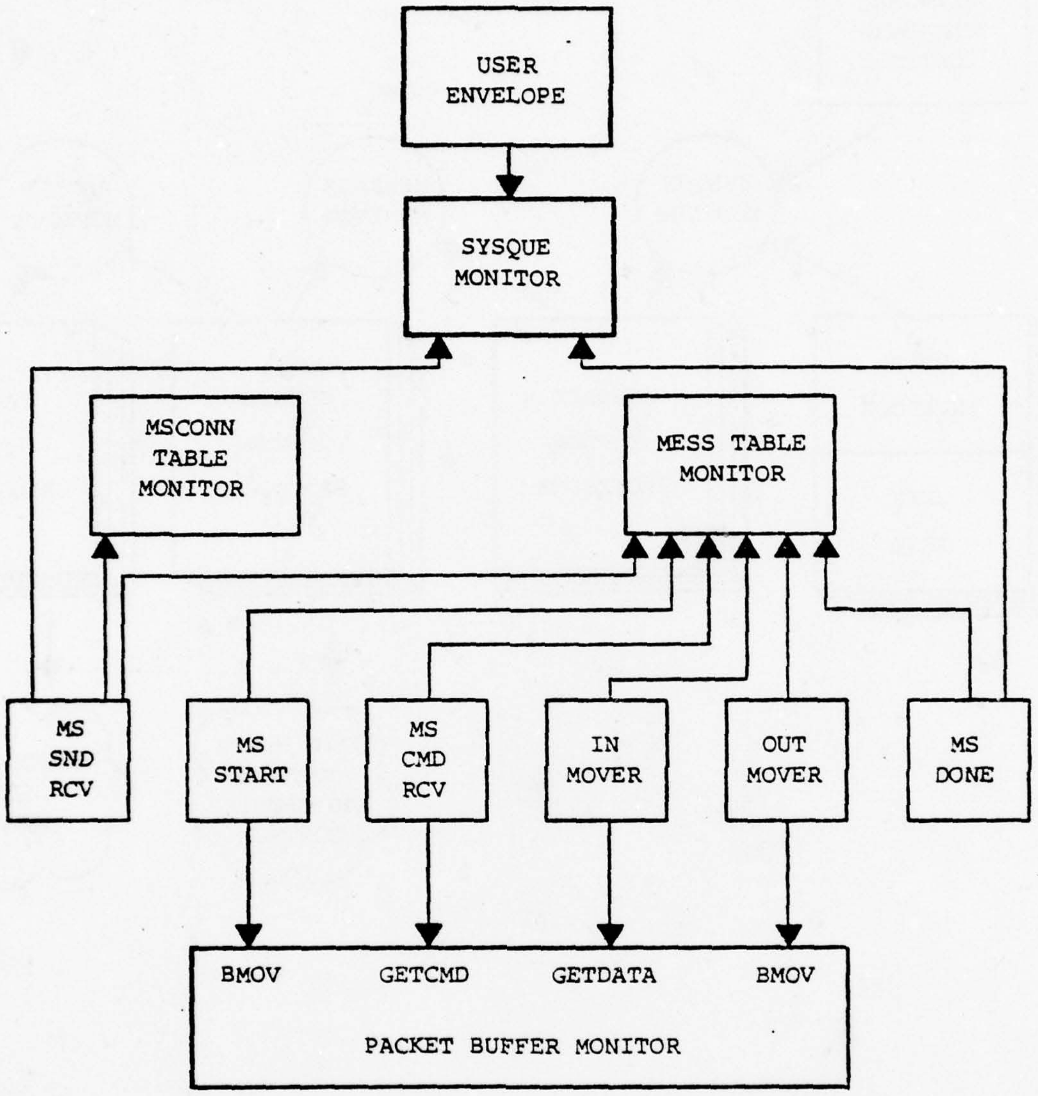


Figure 4.4

Message System Access Graph

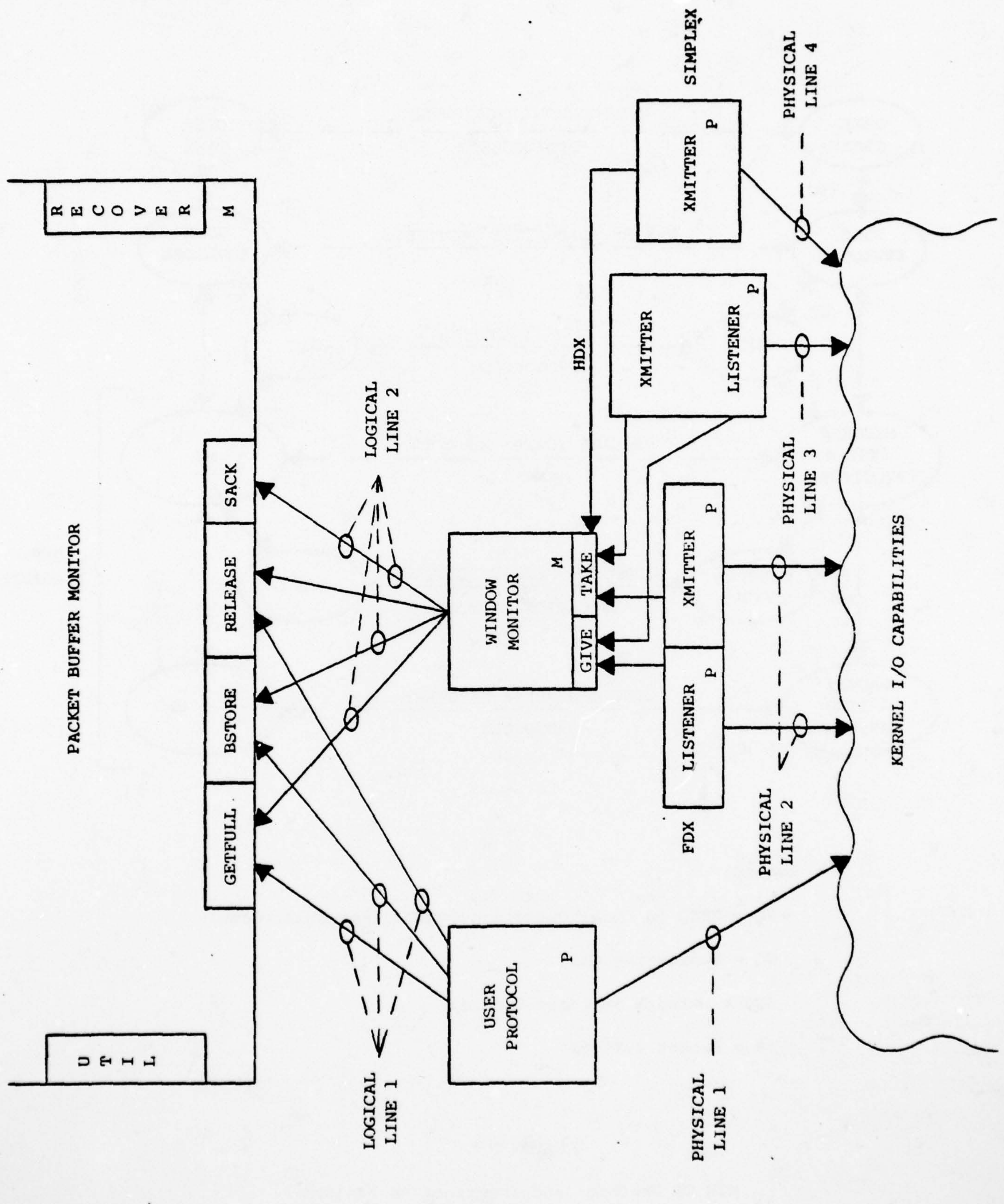
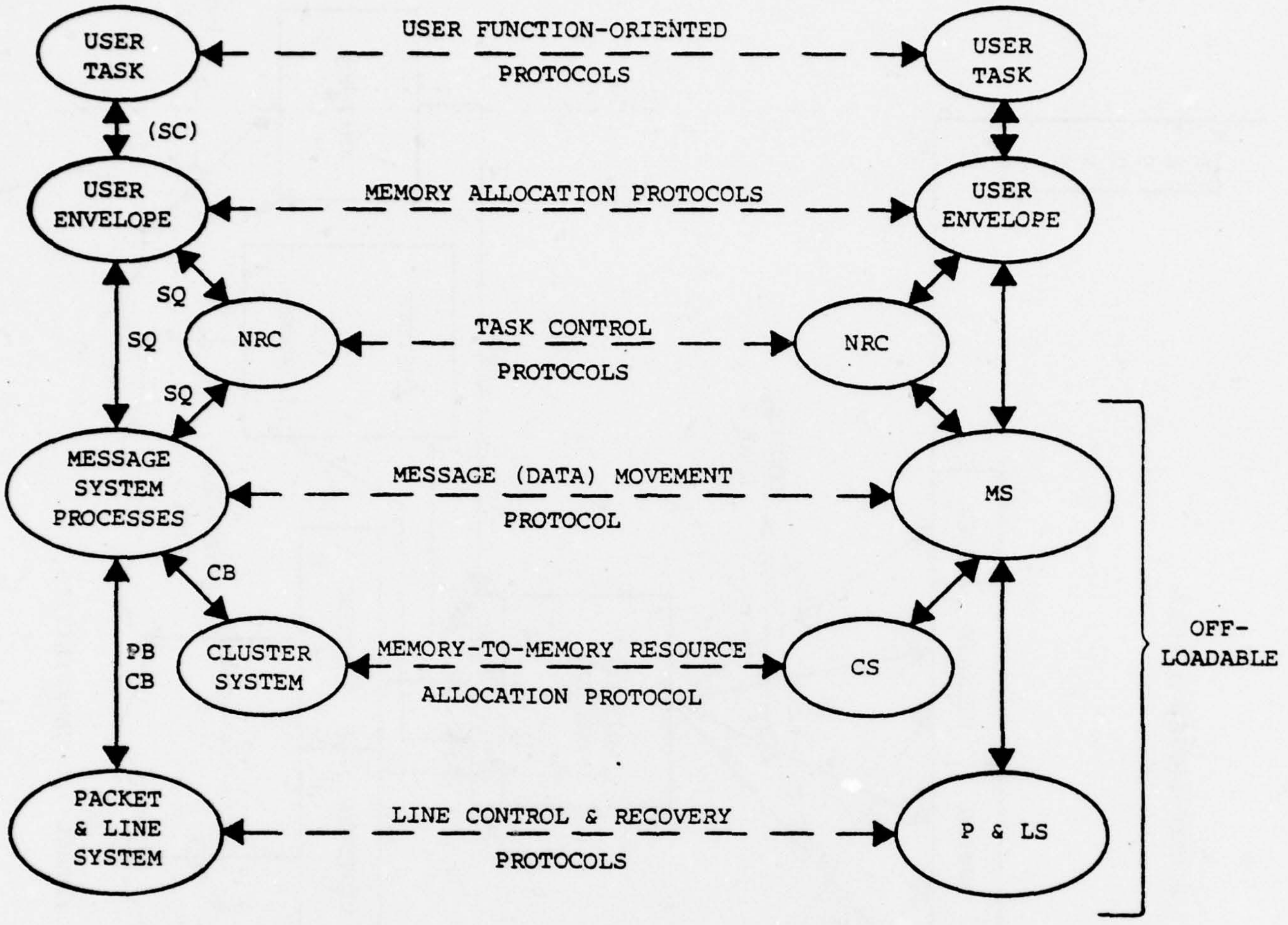


Figure 4.5

Packet System Access Graph



USER TASK
USER FUNCTION-ORIENTED PROTOCOLS
USER TASK
(SC)
USER ENVELOPE
MEMORY ALLOCATION PROTOCOLS
USER ENVELOPE
SQ
SQ
NRC
TASK CONTROL PROTOCOLS
NRC
SQ
MESSAGE SYSTEM PROCESSES
MESSAGE (DATA) MOVEMENT PROTOCOL
MS
CB
PB
CB
CLUSTER SYSTEM
MEMORY-TO-MEMORY RESOURCE ALLOCATION PROTOCOL
CS
OFF-LOADABLE
PACKET & LINE SYSTEM
LINE CONTROL & RECOVERY PROTOCOLS
P & LS

Legend:

SQ = SYSQ in Local Operating System (Assembler Code)

SC = Subroutine Call

NRC = Network Resource Control

PB = Packet Buffers

Figure 4.6A

MIMICS Protocol and Interface Mechanisms

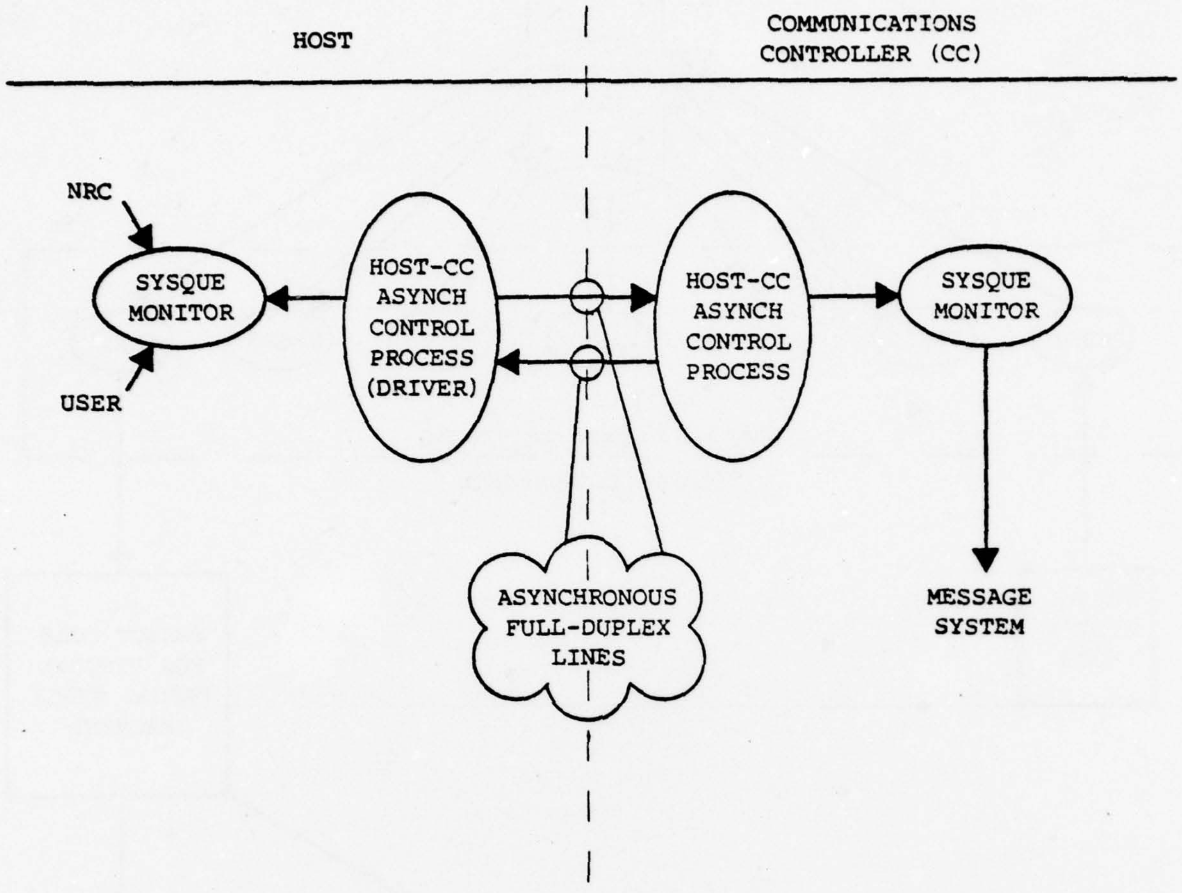

HOST
COMMUNICATIONS CONTROLLER (CC)
NRC
USER
SYSQUE MONITOR
HOST-CC ASYNCH CONTROL PROCESS (DRIVER)
HOST-CC ASYNCH CONTROL PROCESS
SYSQUE MONITOR
ASYNCHRONOUS FULL-DUPLEX LINES
MESSAGE SYSTEM

Figure 4.6B

Implementation of SYSQUE Between Host And Communications Controller

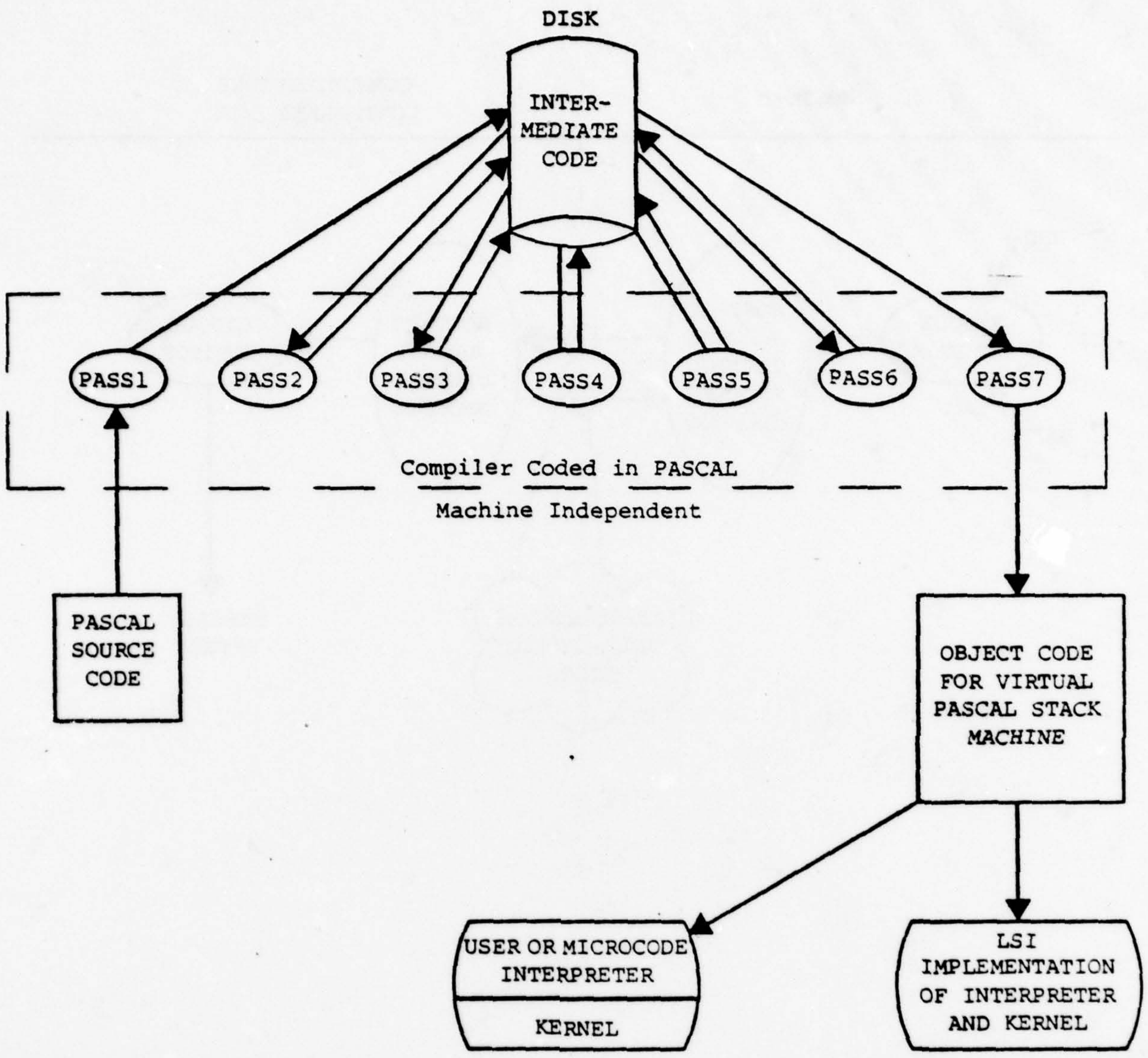


Figure 4.7

Portability of Concurrent PASCAL Using
The Interpretation Technique

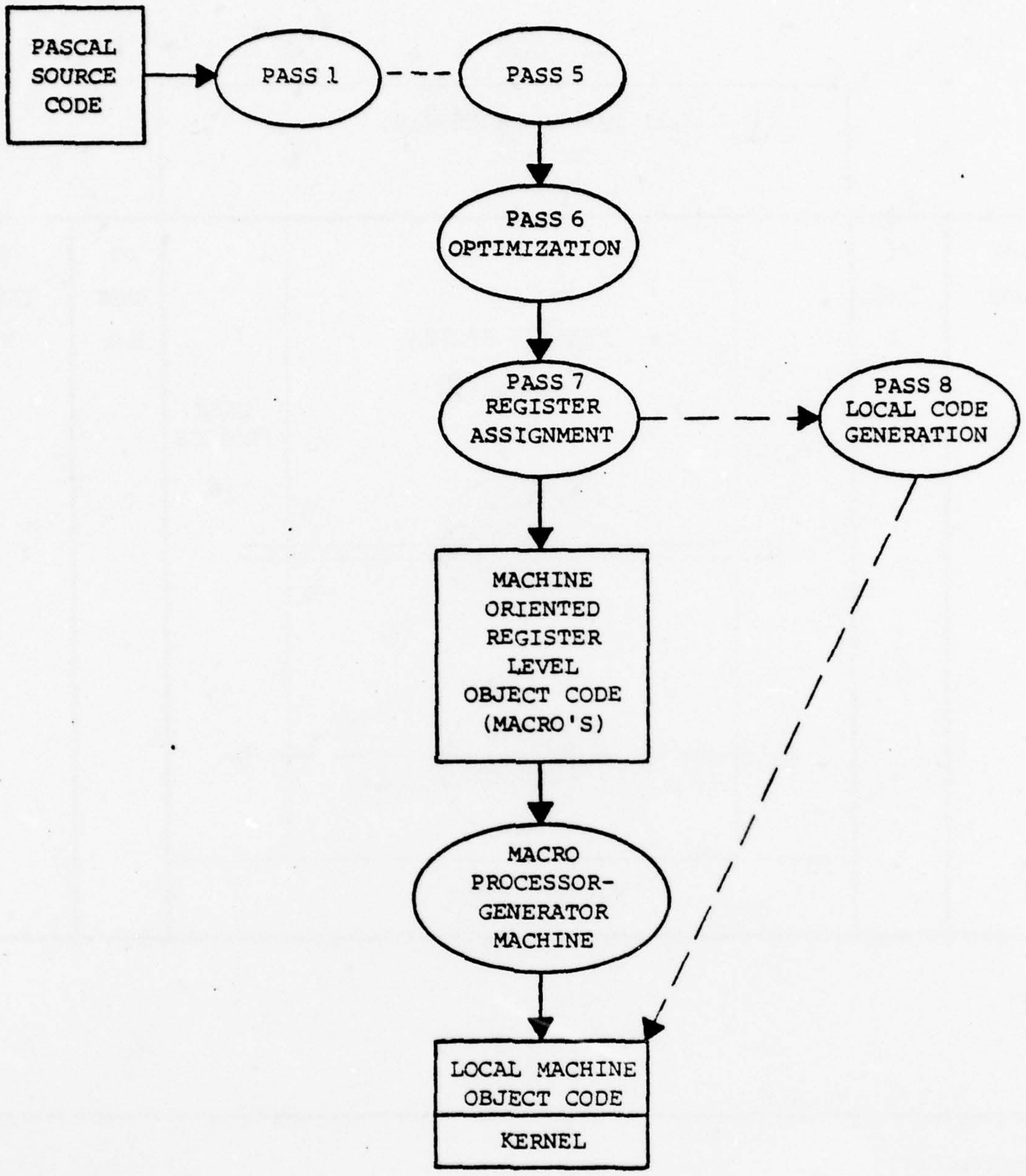

PASCAL SOURCE CODE
PASS 1
PASS 5
PASS 6 OPTIMIZATION
PASS 7 REGISTER ASSIGNMENT
PASS 8 LOCAL CODE GENERATION
MACHINE ORIENTED REGISTER LEVEL OBJECT CODE (MACRO'S)
MACRO PROCESSOR-GENERATOR MACHINE
LOCAL MACHINE OBJECT CODE
KERNEL

Figure 4.8

Portability of Concurrent Pascal Using Compiled Code

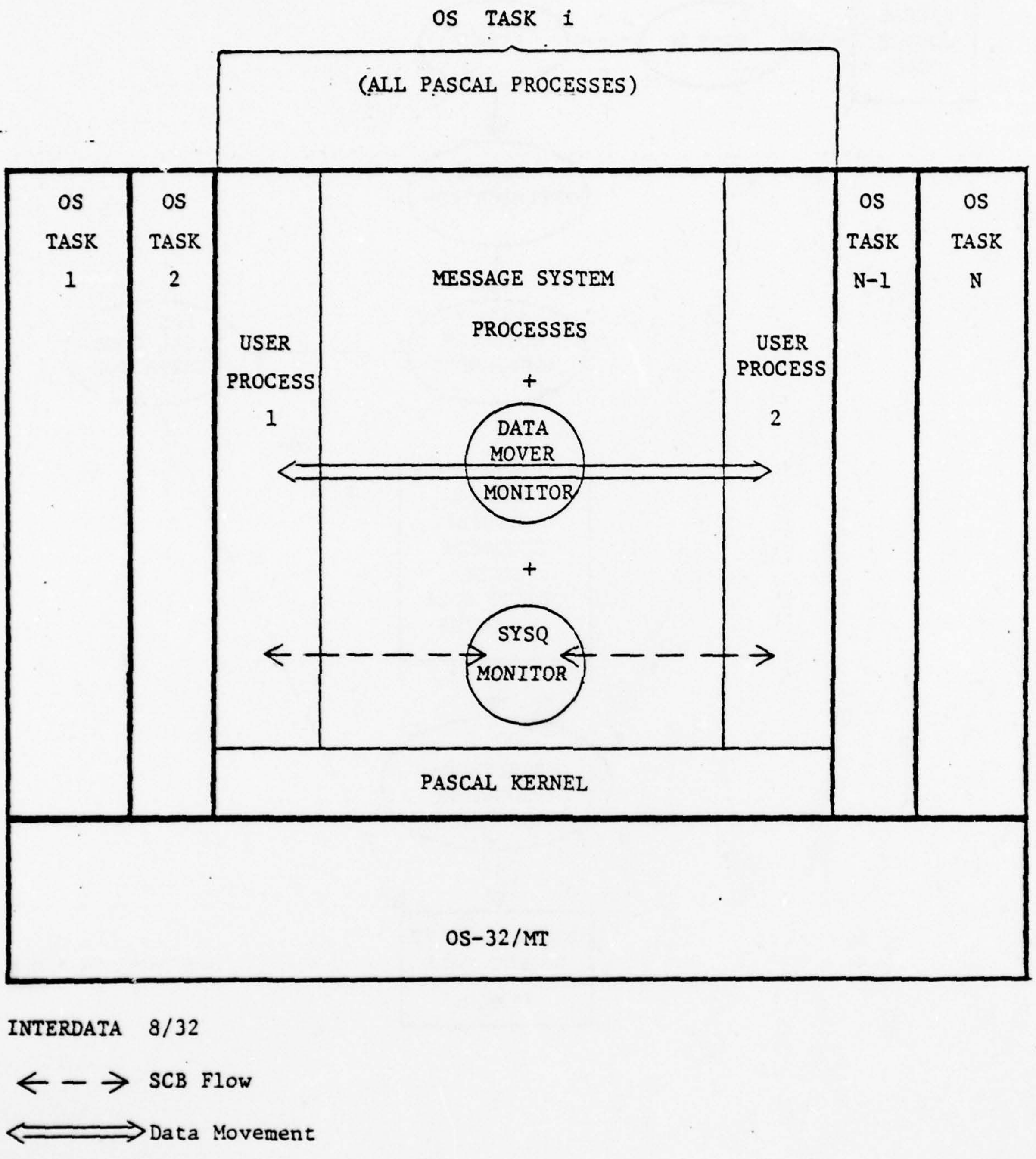
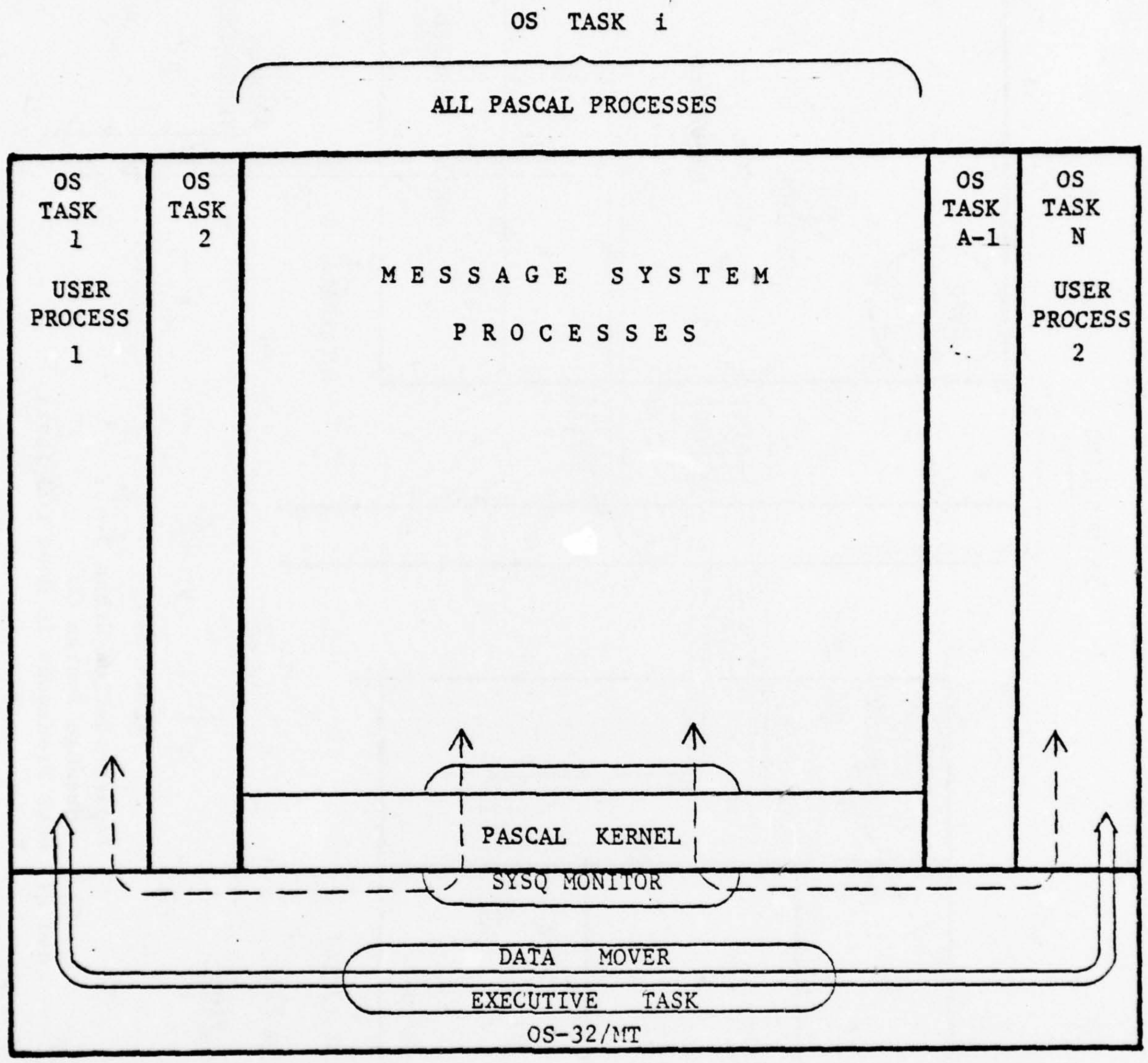


Figure 5.1

Implementation Stage 1--
All Message System and User Processes
in One LOS Task--All Code in PASCAL
(Simulate local, cluster, and remote communications)



INTERDATA 8/32

<-- --> SCB Flow

<==> Data Flow

Figure 5.2

Implementation Stage 2.1--
Message System Processes in one LOS Task
and User Processes in Other LOS Tasks

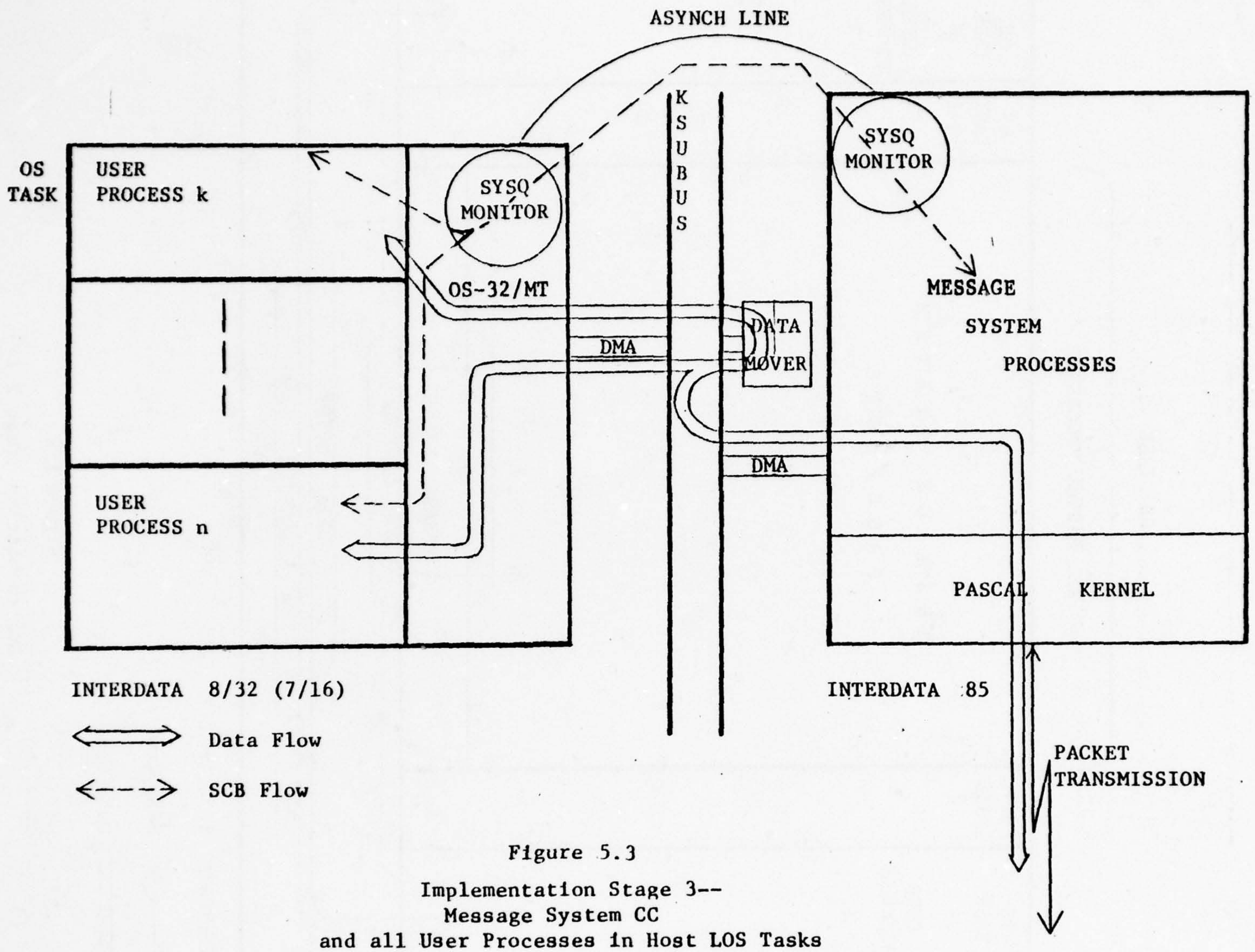


Figure 5.3

Implementation Stage 3--
Message System CC
and all User Processes in Host LOS Tasks

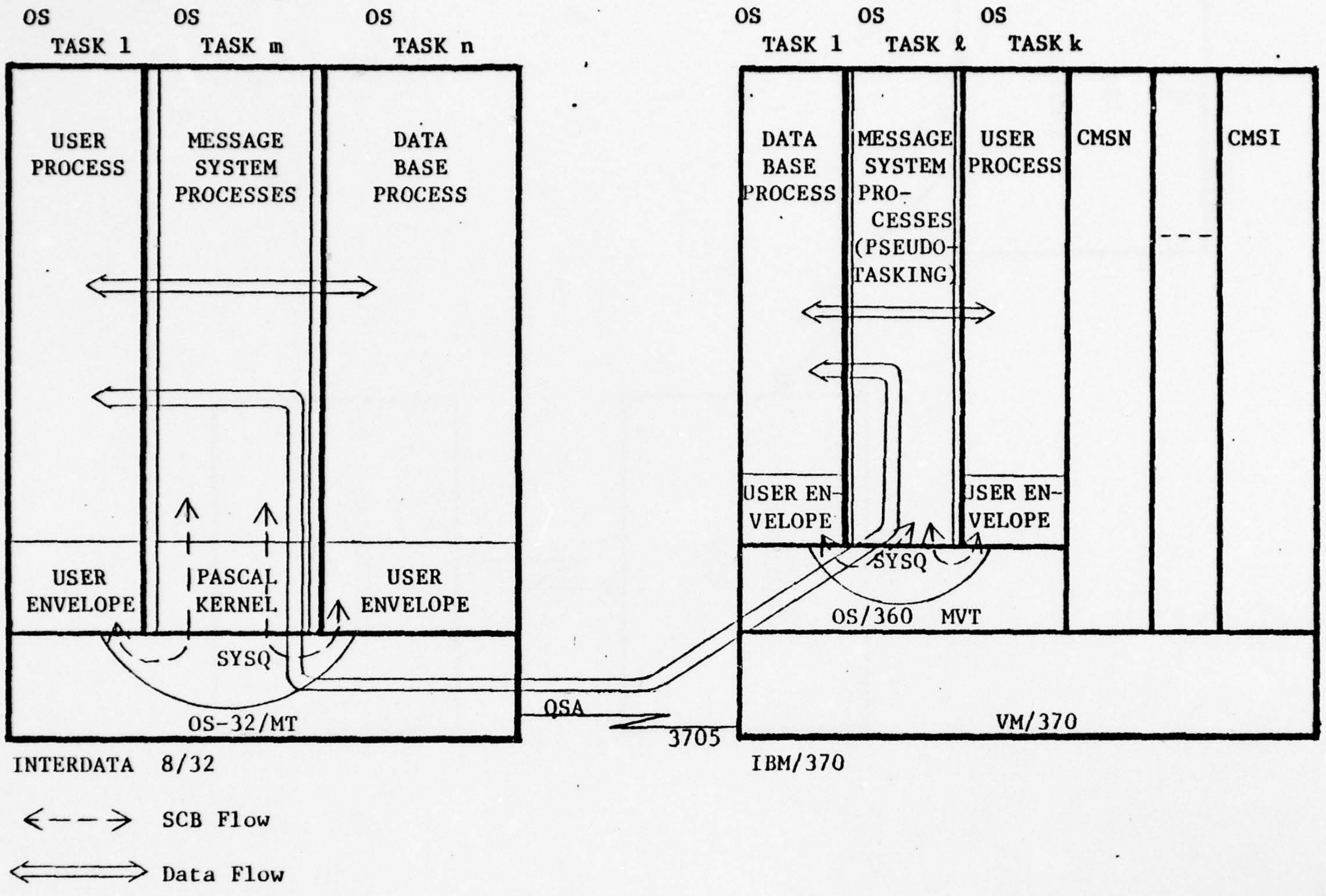


Figure 5.4

Implementation Stage 2.2--
Host Task to Host Task

AD-A052 752 KANSAS STATE UNIV MANHATTAN DEPT OF COMPUTER SCIENCE F/G 17/2
PROJECT REPORT FOR FUNCTIONALLY DISTRIBUTED COMPUTER SYSTEMS DE--ETC(U)
FEB 78 V E WALLENTINE DAAG29-76-G-0108
UNCLASSIFIED ARO-13835.1-A-EL-PT-2 NL

2 OF 2
AD
A052752

END
DATE
FILMED
5-78
DDC

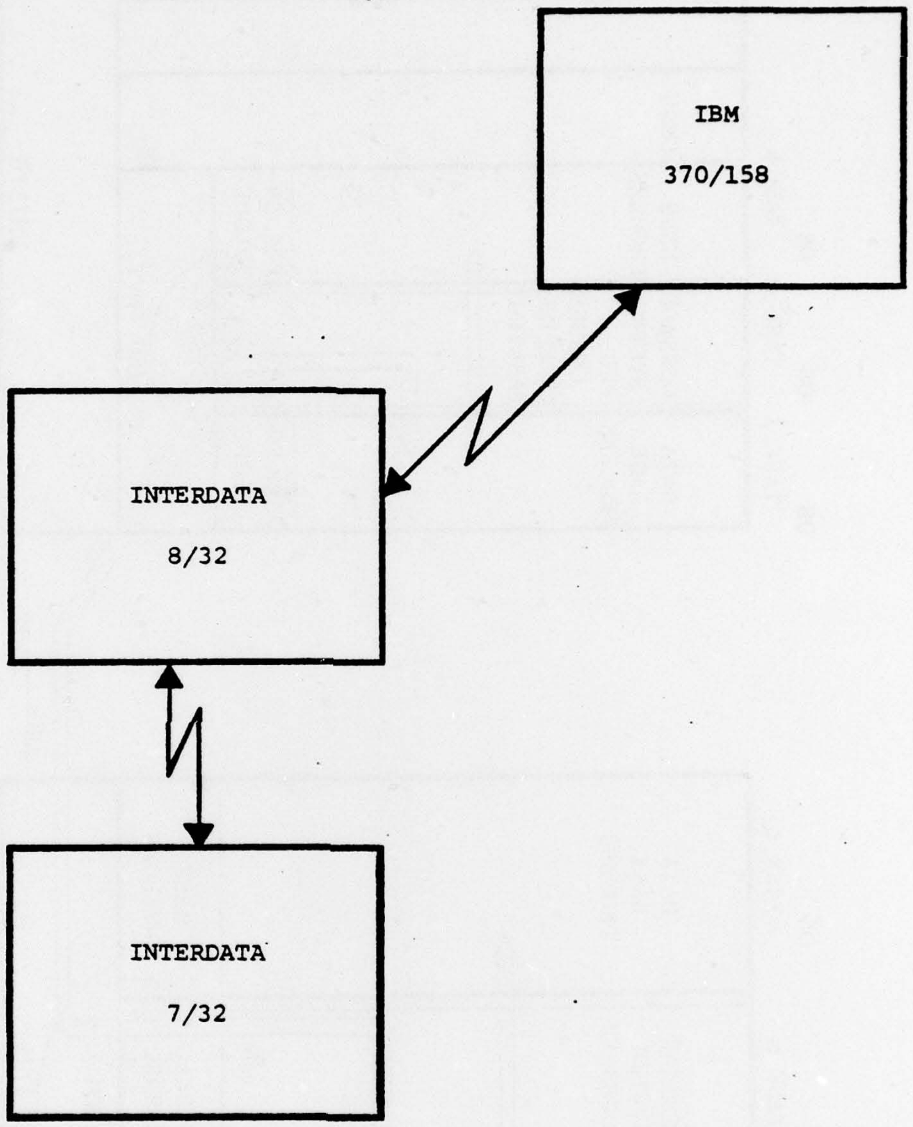


Figure 6.1

Data Base Network Topology

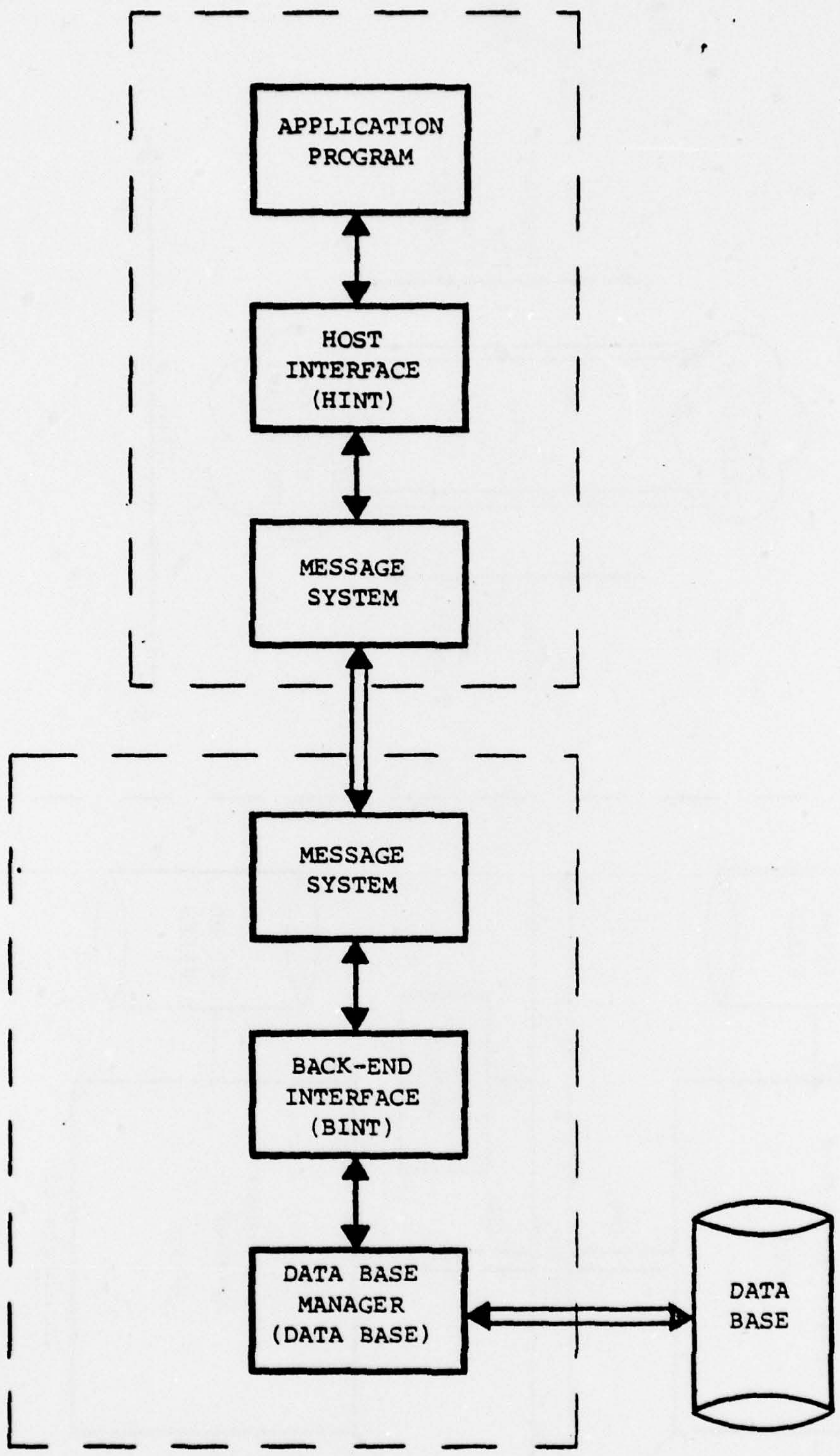
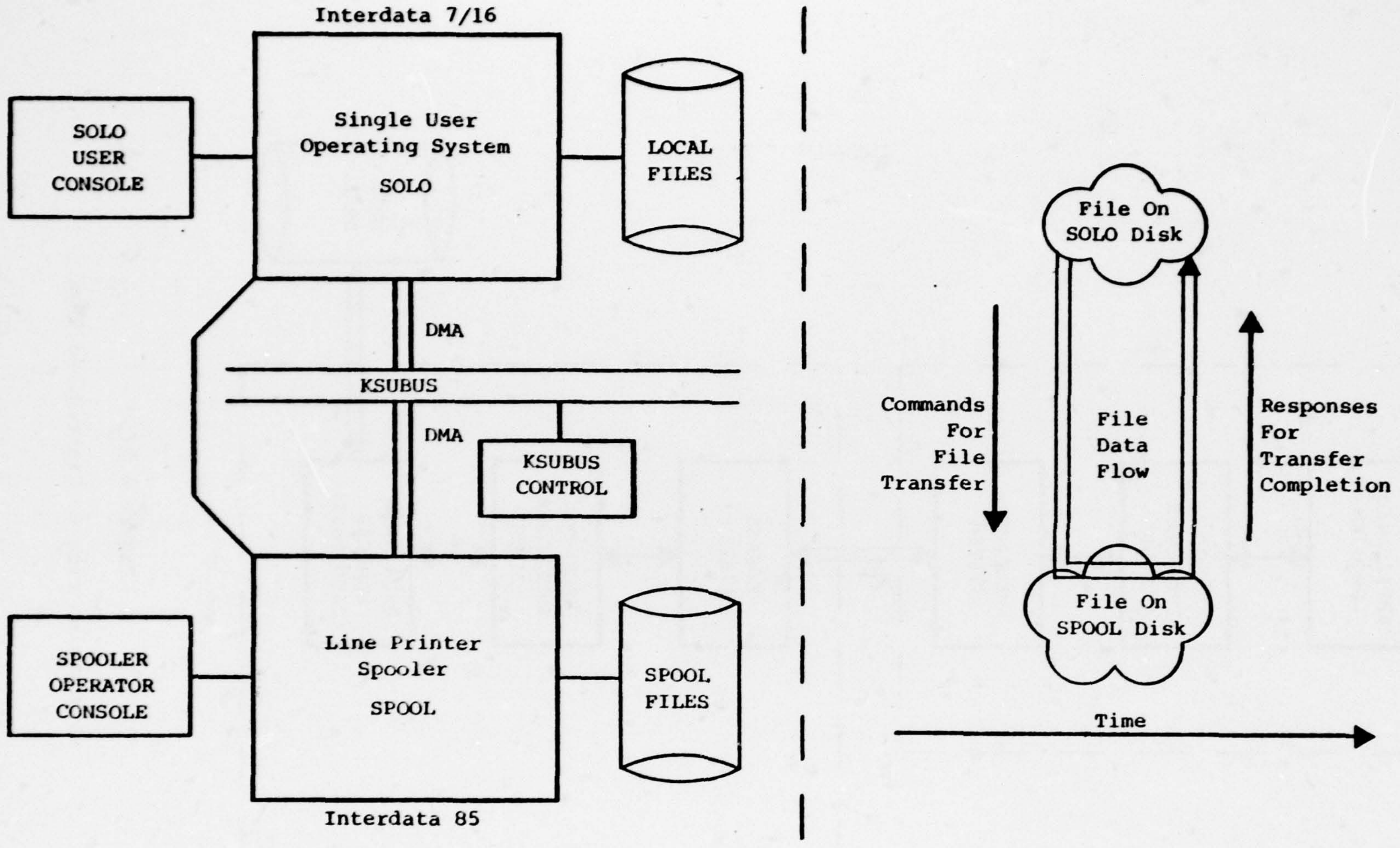


Figure 6.2

Information Flow in Distributed DBMS



a) KSUBUS Demonstration System

b) Data Flow in File Transfer Protocol

Figure 6.3

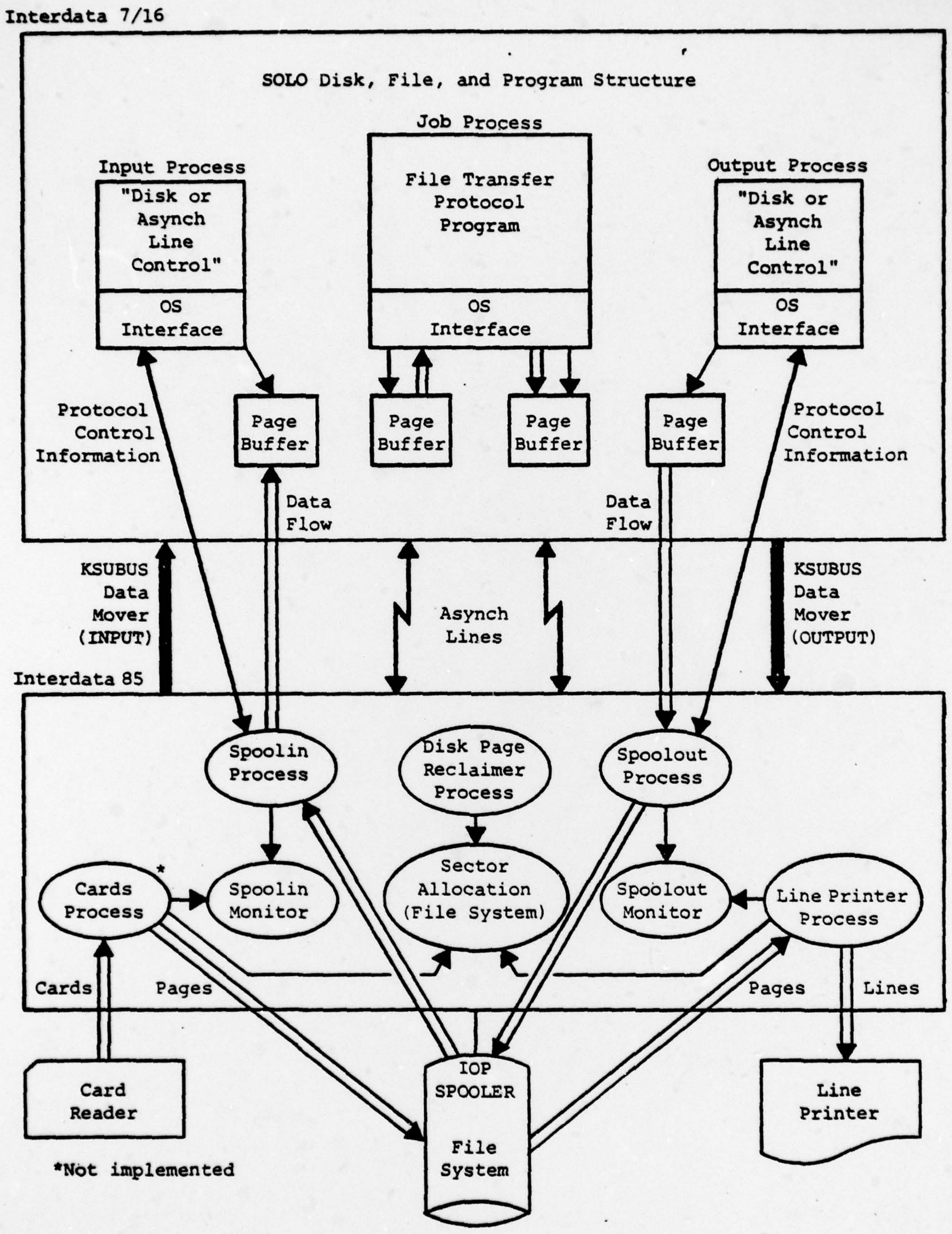


Figure 6.4

Distributed Spooler Operating System and Structure